TUM School of Computation, Information and Technology
Technische Universität München

Design and Evaluation of Demand- and Topology Reconfiguration-aware Networks

Johannes P. D. Zerwas, M.Sc.

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Die Dissertation wurde am 26.06.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 19.10.2023 angenommen.

To my darling wife.

Abstract

The digitalization of our society has led to the adoption of novel data-centric applications and use cases that rely on communication networks. In turn, networks and their operators face new challenges with respect to traffic growth, diversity of communication patterns, and temporal variability of communication demand. Besides the softwarization of the networks through control and data plane programmability, the programmability and reconfiguration of the topology have emerged as one promising approach to render network operation more efficient. Optical technologies allow reconfiguring the topology, i.e., changing the connectivity between networking devices at run-time. However, this results in new problems and challenges for network management. For instance, every reconfiguration interrupts connectivity, which in turn puts stress on the network's control and data plane.

This thesis investigates topological reconfiguration-awareness in demand-aware networks on a macroscopic level by exploring the characteristics of topological reconfigurability in communication networks along three dimensions: reconfiguration delay, networking layers involved in reconfigurations, and reconfiguration classes.

The reconfiguration delay, i.e., the length of the interruption due to reconfiguration, directly impacts the network operation efficiency. It can vary widely depending on the equipment used. In addition to the optical switching devices, also the link endpoints contribute to this delay. In order to investigate this first characteristic and provide a better understanding of how commercially off-the-shelf (COTS) networking devices behave under reconfigurations, this thesis presents procedures to measure the end-to-end reconfiguration delay on the control and data plane. It also analyzes the results of a measurement campaign of COTS networking devices and derives statistical models of the reconfiguration delay. The results show, a strongly varying behavior across the devices. Finally, the thesis introduces a flexible emulation framework that aids the evaluation of different reconfigurable topologies using only COTS equipment.

The second dimension is motivated by the fact that topological reconfigurations unavoidably lead to reconfigurations on higher networking layers, e.g., the IP routing. The question arises if and how these reconfigurations can be jointly optimized with the topological reconfigurations. This thesis assesses the benefits of such joint optimization and reconfiguration of multi-layer networks. It provides a multi-layer modeling and corresponding mixed integer program for a cooperative environment between an Internet Service Provider (ISP) and content providers (CPs). In contrast to previous work, the proposed joint optimization spans the IP topology, IP routing, and the demand layer. The evaluation on production data from a large ISP demonstrates its benefits across a range of reconfiguration frequencies, achieving 15% reduction in deployed capacity on average.

As far as the reconfiguration classes are concerned, there are two classes: demand-aware (*DA*) and demand-oblivious (*DO*). Utilizing their potential requires agility of the routing algorithms and control planes. This is particularly the case for datacenter network (DCN) environments with frequent reconfigurations. As the timely propagation of an up-to-date view of the network is challenging, many designs fall back on separating the traffic on the dynamic (adaptive) topology part from that on the static part, creating a situation of segregated routing. This thesis proposes a novel, high-throughput reconfigurable datacenter network (RDCN) design, coined Duo, which builds on the properties of de Bruijn graphs. These graphs allow greedy routing along the shortest paths using local information. The property translates to the dynamic case facilitating the design of an efficient control plane. Duo can be implemented using longest prefix matching and provides out-of-the-box support for TCP and other transport protocols. The evaluation results demonstrate Duo's superiority over state-of-the-art solutions, and a proof-of-concept implementation shows its feasibility.

The two reconfiguration classes have shown their benefits for different traffic patterns. Choosing the best-suited reconfiguration class is particularly challenging, as the traffic may be a mix of several patterns. This thesis proposes two systems that integrate both reconfiguration classes into a single topology and, thereby, introduces macroscopic awareness for the third characteristic of topological reconfigurations. The proposed systems Cerberus and Trio contain three sub-topologies that are beneficial for different traffic patterns. The sub-topologies are sized to match the shares of the traffic classes in the overall traffic mix. Cerberus and Trio utilize flow sizes or application-level information to assign traffic to the best-suited sub-topology. In addition, using label-based source routing and link endpoint-based management of the sub-topologies' reconfiguration classes, Trio also features the ability to adapt the sizes of the sub-topologies. This enables Trio to continue matching the traffic even if the traffic mix changes – a new dimension of demand-awareness. The results show that both concepts outperform state-of-the-art solutions and illustrate the benefits of matching the reconfiguration classes in the topology to the traffic mix in terms of increased throughput and lower flow completion times.

Kurzfassung

Die Digitalisierung unserer Gesellschaft hat zur Einführung neuer datengestützter und auf Kommunikationsnetzen basierenden Anwendungen und Anwendungsfällen geführt. In Folge dessen sind Netze und ihre Betreiber mit neuen Herausforderungen in Bezug auf Verkehrswachstum, Vielfalt der Kommunikationsmuster und zeitlicher Variation des Kommunikationsbedarfs konfrontiert. Neben der Softwarisierung der Netzwerke auf der Steuerungs- und Datenebenen, haben sich die auch die Programmierbarkeit und Rekonfiguration der Netztopologie als vielversprechende Ansätze herausgestellt, um die Effizienz des Netzbetrieb zu erhöhen. Optische Technologien ermöglichen es, auch in kabelgebundenen Netzen, die Netztopologie zu verändern, das heißt, die Verbindungen zwischen den Netzelementen zur Laufzeit anzupassen. Dieses Anpassen der Verbindungen auf der physikalischen Ebene führt zu neuen Problemen und Herausforderungen für die Verwaltung und den Betrieb der Netzwerke. Jede Rekonfiguration der Netztopologie unterbricht Verbindungen auf höheren Netzebenen und führt dadurch zu Belastungen auf Steuerungs- und Datenebenen.

Diese Dissertation untersucht die Berücksichtigung der topologischen Rekonfiguration auf einer makroskopischen Ebene im Netzmanagement. Dafür werden die Charakteristiken solcher Rekonfigurationen entlang drei Dimensionen näher betrachtet: der Rekonfigurationsdauer, der involvierten Netzwerkschichten sowie der verwendeten Rekonfigurationsklassen.

Die Rekonfigurationsdauer, das heißt, die Länge der Verbindungsunterbrechung aufgrund der topologischen Rekonfiguration, hat einen direkten Einfluss auf die Effizienz des Netzwerkbetriebs. Sie kann in Abhängigkeit der verwendeten Geräte stark variieren. Neben dem optischen Schaltelement tragen auch die Verbindungsendpunkte zur Rekonfigurationsdauer bei. Um diese erste Charakteristik genauer zu untersuchen und ein besseres Verständis zu bekommen, wie sich kommerziell erhältliche Netzwerkgeräte in Anbetracht topologischer Rekonfigurationen verhalten, wird in dieser Dissertations zunächst eine Messmethode um die Ende-zu-Ende Rekonfigurationsdauer auf der Steuerungs- und Datenebene zu ermitteln, präsentiert. Im Anschluss werden die Ergebnisse einer Messkampagne von verschiedenen kommerziell erhältlichen Netzgeräten anlysiert und statistische Modelle der Rekonfigurationsdauer aus den Messdaten hergeleitet. Die Ergebnisse zeigen ein stark unterschiedliches Verhalten der untersuchten Geräte. Im Anschluss wird eine flexible Emulationsumgebung, die die Bewertung verschiedener rekonfigurierbarer Topologien ermöglicht und dafür nur kommerziell verfügbares Equipment benötigt, präsentiert.

Die zweite Dimension ist aus der Tatsache begründet, dass Rekonfigurationen der Netztopologie zwangsläufig zu Rekonfigurationen auf höheren Netzwerkschichten führen, zum Beispiel dem IP Routing. Es stellt sich die Frage, ob und wie diese Rekonfigurationen zusammen mit Rekonfigurationen der Netztopologie optimiert werden können. Dafür werden die Vorteile einer solchen gemeinsamen Optimierung und Rekonfiguration von mehrschichtigen Netzwerken untersucht. Eine neu-formulierte, mehrschichtige Modellierung und ein dazugehöriges Mixed Integer Program zur Optimierung in einer kooperativen Umgebung aus einem Internetserviceanbieter und Inhalteanbietern bilden die Grundlage dafür. Anders als vorherige Arbeiten, erstreckt sich die vorgestellte gemeinsame Optimierung über die Ebenen der IP Topologie, das IP Routing und die Kommunikationsbedarfsschicht. Die Auswertung auf Basis der Daten eines großen Internetserviceanbieters verdeutlichen die Vorteile des Ansatzes über verschiedene Rekonfigurationshäufigkeiten, wodurch eine Reduktion der benötigten Kapazität um 15% im Mittel erreicht wird.

Bei den Rekonfigurationen der Netzwerktopologie haben sich zwei Klassen herauskristallisiert: bedarfsorientiert und bedarfsagnostisch. Um ihr gesamtes Potenzial auszu-schöpfen müssen Routingalgorithmen und die Steuerungsebene schnell reagieren können. Dies ist besonders in Rechenzentrumsnetzwerken mit häufigen Rekonfigurationen der Fall. Da eine ausreichend zügige Verteilung des aktuellen Netzwerkzustandes durch das gesamte Netz oftmals schwer zu erreichen ist, greifen viele existierende Lösungen darauf zurück, den Verkehr auf dem dynamischen Teil der Topologie von dem auf dem festen Teil der Topologie zu trennen. Diese Dissertation führt ein neuartiges Design für einen hohen Netzdurchsatz ein. Das System DUO baut auf den Eigenschaften von de Bruijn Graphen auf. Diese Graphen ermöglichen ein gieriges Routing über den kürzesten Pfad zum Ziel nur mit Hilfe von lokal verfügbaren Informationen. Diese Eigenschaften übertragen sich auch auf den Fall einer dynamischen Erweiterung des Graphen und ermöglichen dadurch eine effiziente Realisierung der Steuerungsebene. DUO kann mit Longest Prefix Matching implementiert werden und unterstützt TCP und andere Transportprotokolle. Die Auswertung zeigt DUOS Überlegenheit gegenüber den Lösungen des Stand der Technik und eine Laborimplementierung zeigt die Machbarkeit des Konzepts.

Die beiden Rekonfigurationsklassen haben Vorteile für unterschiedliche Verkehrsmuster. Die passende Rekonfigurationsklasse auszuwählen ist besonders herausfordernd, da der Gesamtverkehr auch eine Mischung verschiedener Muster sein kann. Diese Dissertation führt zwei Systeme ein, die beide Rekonfigurationsklassen in einer Topologie vereinen. Die beiden Systeme, CERBERUS und TRIO, haben drei Teiltopologien, die jeweils für unterschiedliche Verkehrsmuster geeignet sind. Die Größen der Teiltopologien sind an die Anteile der Verkehrsmuster am Gesamtverkehr angepasst. CERBERUS und TRIO verwenden die Flussgrößen oder Anwendungsinformationen um den Verkehr der besten Teiltopologie zuzuweisen. Zusätzlich ermöglichen zwei Designentscheidungen von TRIO, ein labelbasiertes Quellenrouting sowie eine endpunktbasierte Verwaltung der Teiltopologien, sodass die Grö"sen der Teiltopologien über Zeit angepasst werden können. Dadurch kann TRIO sich dem Verkehr anpassen, auch wenn sich die Verkehrsmischung verändert. Dies bedeutet eine neue Dimension der Bedarfsorientierung. Die Auswertungsergebnisse zeigen, dass beide Systeme den Stand der Technik überbieten. Gleichzeitig verdeutlicht dies die Vorteile, die durch das Anpassen der Rekonfigurationsklassen an die Verkehrsmischung in Form von höherem Netzdurchsatz und niedrigeren Flussübertragungszeiten entstehen.

Contents

Chapter 1

Introduction

Today's communication networks have become more and more the backbone of our digitialized society. Next-generation communication technologies such as 5G and 6G networks have led to a substantial increase in available bandwidth and, thereby, have enabled the adoption of new data-centric applications [28, 29, 30]. Examples are video and game streaming, video broadcasting, smart assistants, ubiquitous data collection and analytics, or the metaverse. This development challenges network operators who strive toward efficient network operation along three dimensions:

First, we observed massive traffic growth in the past years. The average and the peak traffic rates have steadily increased with a compound average growth rate of 30% between 2018 and 2022 [31]. Second, besides the traffic growth, the applications' patterns and the requirements from the applications toward the network have changed too. Fostered by the shift to remote work during the COVID-19 pandemic, users upload or stream videos from home [32]. Moreover, the amount of machine-to-machine type communication increases [33]. Looking at the different applications that share the network, the requirements range from low bandwidth but latency-sensitive traffic, e.g., from Tactile Internet or control loops in industrial networks towards bandwidth-hungry but latency-tolerant applications, e.g., from large file transfers [30]. In particular in datacenters (DCs) that host cloud environments, a plethora of applications are running over the same network. The diversity ranges from web frontends via application backends and databases to distributed data analytics and machine learning (ML) traffic, all on the same network [34, 35, 36]. A recent study by Avin et al. [37] illustrates that all these applications result in different traffic patterns in the network – both in the temporal and spatial domains. Third, the temporal variability of the traffic increased. That is, the bursty user or application behaviors put more stress on the network in the case of bandwidth-hungry applications such as large software updates or ML [38, 39].

To address these problems, software defined networking (SDN) has been introduced as a paradigm to increase the flexibility of communication networks [40]. With SDN, operators can program and adapt the network at runtime. Putting the initial focus of SDN to the control [40] and data plane [41] programmability, the underlying network topology may still limit the gains from SDN. Standard topology designs are static and often oblivious to traffic in the network [42, 43, 44]. In particular, the increased temporal variability of demand patterns has raised the question if there is still a single best network topology that operators should choose. Thus, the SDN paradigm has recently been extended to the network topology [45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56]. Thanks to the advances in optical technologies, *reconfigurable topologies* have been adopted to drive the flexibility of wide area networks (WANs) and datacenter networks (DCNs) and help operators to sustain a high quality of service (QoS). Reconfigurable topologies build on an optical infrastructure that interconnects a set of network nodes. Each node is equipped with a number of sending and receiving ports that enable parallel (logical) connectivity to other nodes. These logical connections can be adapted over time by leveraging recent optical technologies. For instance, a node is first logically connected to another node and later to a third node. The used optical technology and infrastructure restrict the number and set of possible connections. With this, the topology can adapt to the demand and provide high bandwidth between nodes where needed. However, while this reconfigurability, in general, has the potential to adapt the topology to the present demand pattern, i.e., make the topology *demand-aware (*DA*)*, it comes with three operational problems:

First, the reconfiguration introduces an interruption of the network connectivity. Intuitively, a link is not available when a reconfiguration happens. During this time, the network's capacity deteriorates. The length and frequency of reconfigurations impact the overall efficiency of the network operation. In addition, varying technologies and network devices might behave differently under or due to such topological reconfigurations. Thus, a solid understanding of the reconfiguration behavior is vital to network operators. While this downside of topological reconfigurations is generally acknowledged in the literature, a thorough investigation of different influencing factors is missing.

Second, topological reconfigurations trigger reconfigurations on higher layers, such as in the IP routing and traffic engineering (TE) or even on the application layer, e.g., in L7-load balancing. While it is obvious that the endpoints of a reconfigured link must update their forwarding information, updates on other nodes might be necessary, too, in order to leverage the full potential of the adapted topology.

Third, different classes of reconfigurations have emerged in the literature: demand-oblivious (*DO*) and *DA* reconfigurations. Operators face the task of selecting the right class for their network. The former does not consider the demand in the network, i.e., it follows a pre-defined schedule and therefore, can generally reconfigure faster (there is no need to collect demand information). However, the applied topologies might be suboptimal. In contrast, *DA* reconfigurations allow operators to optimize the topology according to a specified metric at the cost of a lower reconfiguration frequency.

The overall goal of this doctoral thesis is to present designs of demand-aware networks that consider the characteristics of topological reconfigurations on a macroscopic level. To this end, the thesis addresses three sub-goals that align with the operational problems presented above. Accordingly, the major contributions of this thesis are three-fold: the measurement and modeling of topological reconfigurations, the design of a *DA* network that considers the impact of reconfigurability on multiple layers, and the design of a *DA* network that considers the different flavors of reconfigurability combined with an efficient control plane. The following sections describe these research challenges and the thesis' contributions in more detail.

1.1 Research Problems & Challenges

This section describes the main research problems and challenges that are addressed in the subsequent Chapters 4-7.

1.1.1 Modeling and Evaluation of Topological Reconfigurability

Intuitively, reconfigurations of the network topology stress the network since the components must adapt to a new state. For instance, network interface cards (NICs) must establish new links and propagate this new state to routing daemons or network controllers. These, in turn, need to update their view of the network topology to take necessary actions, e.g., to update their forwarding tables.

Reconfiguration times can vary widely depending on the use case and equipment. For instance, in WANs, links established with common bandwidth-tunable optical transceivers or reconfigurable optical add-drop multiplexers (ROADMs) have reconfiguration times in the order of minutes [57, 58]. There is sufficient time for other network equipment to adapt to the new state on this timescale. For DCNs, there exists optical hardware with reconfiguration times in the order of micro-seconds (e.g., Mordia [52], RotorNet [48]) or even nano-seconds (e.g., Sirius [53] or PULSE [59]). However, many commercially off-the-shelf (COTS) components in DCN, such as NICs or packet switches, are not designed with frequent reconfigurations or state changes in mind. Consequently, they might behave unexpectedly and, in the worst case, hinder the efficient operation of a reconfigurable datacenter network (RDCN).

Thus, a first challenge on the way toward more efficient demand-aware topologies is a deeper understanding of the components' behavior under reconfigurations. The state-of-the-art provides individual evaluations of COTS equipment, e.g., [54, 51, 47, 48]. However, a thorough exploration of whether and how existing programmable networking equipment can cope with such reconfigurations is missing. Such a benchmark should cover different devices and varying reconfiguration scenarios. Moreover, data sheets of optical, reconfigurable equipment provide measurements for the physical layer. Nevertheless, to understand the end-to-end reconfiguration behavior, the design of a measurement procedure that considers the end-to-end reconfiguration delay of optical links, i.e., the time until packets are received via the new circuit and the delay on the control plane, is needed.

The second aspect of this challenge is to evaluate and compare system designs with different types of reconfigurable topology components. Such an evaluation is challenging, in particular for experimental assessments. Testbeds of RDCNs are usually based on custom-built prototypes and rely on modified hardware and software which is not publicly available. Therefore, a broad comparison is not easily achievable.

1.1.2 Leveraging Reconfigurations on Multiple Layers in WANs

The design and operation of communication networks are commonly approached in a multi-layer fashion [60]. Topological reconfigurations take place on one of the lower network layers. As a result, these reconfigurations might impact the state of the upper layers. For instance, in a WAN scenario, the first (lowest) layer consists of the logical topology (IP links) mapped to the physical network topology, i.e., the fiber infrastructure. On top, the second layer determines the IP routing (or TE) of the traffic. Changes in the IP topology will require changes in the routing state. Besides changes on the nodes directly affected by the reconfiguration, updates on other nodes might also be needed. For instance, it might be necessary to update the routing tables on other nodes to utilize the available network capacity most efficiently and achieve a high QoS.

It is common practice to optimize and adapt the two mentioned layers in WANs with respect to the observed demand in the network on a coarse timescale [61, 62, 63]. While this is already a challenging optimization problem, it neglects the potential of one additional dimension: optimizing and reconfiguring the demand. Most traffic in end-user-facing WANs, so-called eyeballs, originates from large content providers (CPs) or content delivery networks (CDNs). The large CPs can control to some extent where their traffic enters the end-user-facing WANs, e.g., by modifying the domain name system (DNS) entries of their services. The flexibility from this so-called *end-user mapping* in combination with the reconfiguration of the topology should be leveraged to reduce resource consumption further and increase the QoS for end-users. Such an optimization across three layers relies on different technologies on the optical and IP routing layers, with potentially different reconfiguration properties. Therefore, the evaluation of this joint optimization should capture the individual gains of the layers to provide more comprehensive guidelines to operators.

1.1.3 Design and Implementation of Efficient Control Planes for Demand-aware RDCN

Routing algorithms and control planes for *DA* RDCN can be implemented in a centralized or distributed fashion. The centralized approach, particularly promoted by the SDN paradigm, requires a global, ideally consistent view of the network. Examples of such control planes or DCNs building thereon are given from multiple large operators [64, 65]. However, the global view is already challenging to obtain from a scalability point of view in static network topologies. Given the frequent adaptations of the topology in RDCNs, it becomes even more challenging.

Also, implementing the distributed approach comes with challenges due to the fast reactions needed. The challenge here is not necessarily the algorithmic problem per se but a fast reaction to the new topology state. As mentioned before, to most efficiently utilize newly created links, updates on all nodes in the network are needed. In general, existing RDCNs typically rely on a hybrid topology that combines static and dynamic parts. While such a combination is powerful [47], current architectures support only somewhat restricted routing. First, routing on the dynamic topology part is limited to one or two hops only [47, 66]. Second, routing is often *segregated*, i.e., flows are forwarded on either the static or the dynamic topology part but not both [47, 51]. This segregated routing also comes with additional overheads: during reconfiguration, the links are unavailable, and packets must be buffered. Regular store-and-forward switches are not sufficient to implement this, and more complex buffer management is needed. Thus, there is a need for an RDCN design with a decentralized routing approach that supports routing over multiple hops over both static and dynamic links and relies on local decisions in order to react quickly to changing links.

1.1.4 Design of Macroscopic Topology Reconfiguration-aware Networks for DCs

In the past decade, researchers presented several designs for RDCNs targeting different traffic constellations. Based on their reconfiguration behavior, these designs can be discriminated into two classes: *DA* [66, 47, 51, 45, 46, 54] and *DO* [48, 49, 53]. Static DCNs constitute a third class. Depending on the class, two types of *taxes* (cost) can occur: latency and bandwidth tax. Both taxes impact a flow's performance (e.g., the completion time) depending on the its size. For instance, a high latency tax more strongly impacts small flows (the flow has to wait a long time for a reconfigurable link to come up). In contrast, a large flow (with a large "ideal" completion time) will not experience a substantial increase in its overall completion time. Opposed to this, a high bandwidth tax (the flow utilizes multiple hops in the network) more likely results in congestion, lower throughput, and ultimately, increased completion time for large flows.

As a result, the class of reconfigurable topology that is most suited depends on a flow's size, and the design of a DCN that is aware of the different classes of topological reconfigurability is a challenging task. Given the traffic mix inside the network, the first aspect of the network design is how to dimension the different topology parts to maximize the topology's throughput and minimize the completion times of the flows. The dimensioning should account for the two taxes. The second challenging aspect is designing the architecture and the resource management for such a hybrid topology that consists of several classes of reconfigurable sub-topologies. This challenge includes, on the one hand, an algorithm that optimizes the scheduling for *DA* and *DO* links and, on the other hand, a data plane design and a routing algorithm that enable forwarding across the sub-topologies.

1.2 Contributions

This section overviews and summarizes the contributions of this thesis to the research area of demand-aware, topologically-reconfigurable networks. It briefly describes the conducted research efforts and illustrates their relations. Figure 1.1 visualizes the structure of the thesis with respect to the contributions and methodologies. The major contributions cover two scenarios (WAN and DCN) and align into the following three groups: (1) measurements and evaluation frameworks of reconfigurable topologies, (2) mathematical models, optimization, and numerical analysis of networks using multiple layers of reconfigurations in WANs, and (3) architectures, control and data plane designs of RDCNs that use multiple classes of reconfigurations.

The increased use of data-centric applications has led to new applications and traffic patterns in networks. Some of these new applications can particularly benefit from reconfigurable (and *DA*) networks. As a specific example, the first minor contribution is the analysis of the network traffic of distributed machine learning (DML) frameworks [10]. The study elaborates on network traffic traces from three state-of-the-art DML frameworks, four different ML models, and various configurations. Moreover, it assesses the impact of network parameters. The study demonstrates that the frameworks employ specific traffic patterns depending on the configuration. Therefore, the traffic of such applications is a good candidate for reconfigurable networks.

The second contribution provides a deeper understanding of the behavior of programmable COTS equipment under topological reconfigurations [11]. Therefore, this thesis first provides a meta-analysis of existing measurements of topological reconfigurations to uncover missing aspects for a holistic analysis of the end-to-end reconfiguration delay. It then identifies potential factors of variability for the reconfiguration delay and presents three measurement procedures. They are tailored for two classes of devices: programmable switches and programmable NICs. The measurement procedures do not require specialized measurement equipment as they build on the capabilities of programmable COTS data-plane devices. Assessing six programmable COTS networking devices provides insights into their behavior under optical circuit reconfiguration and verifies the hypotheses about the influence factors. Thereby, it extracts the significant components of end-to-end reconfiguration delay. As a complementation of this contribution, this thesis presents a flexible framework for **Ex**perimentation with **Rec**onfigurable networks

(ExReC) [12]. ExReC can be configured in different ways, allowing us to emulate different RDCN architectures. It relies only on COTS hardware and uses emulation to reduce dependence on hardware that is either expensive or unavailable. The thesis illustrates the versatility and benefits of ExReC under various workloads, including a DML training application.

This thesis' third and first major contribution targets the joint optimization of reconfigurations on multiple layers. To this end, the WAN use case is considered. Multi-layer optimization is a common approach in the design and optimization of WANs. However, existing work is limited to optimizing individual layers or combinations of two. In contrast, this thesis analyzes to what extent and how a reconfigurable optical network topology can be combined with a clever request mapping to optimize the routing of the traffic of large CP or CDNs in an Internet Service Provider (ISP) network [1]. Therefore, it presents a Mixed Integer Program (MIP) formulation that, given the peering locations (point of presence (PoP)) of the CDNs and CPs as well as the end-users' demands, jointly optimizes three layers: CDN user mapping[1], IP routing, and IP topology and its embedding in the optical domain. In order to evaluate the benefits of this joint optimization, the thesis proposes a greedy algorithm that optimizes the network in two stages; CDN user mapping followed by IP routing and topology in the second stage. The proposed approaches are empirically evaluated based on measurement data from the production network of a large ISP. The evaluation shows that the joint optimization can significantly lower the ISP's network loads not only during the critical peak hour but also on average and reduce the required backbone capacity by up to 15%. Besides the gains achieved by the joint optimization, the evaluation features a reconfiguration analysis, elaborating the required amount and frequency of reconfigurations. Moreover, it illustrates the predictability of the required optimization and its benefits during events such as the COVID-19 pandemic or under link failures. Finally, the thesis showcases avenues for a possible system implementation and deployment scenario.

As the fourth and second major contribution, this thesis proposes a novel, cost-effective and *DA* RDCN architecture, Duo which uses static and *DA* network components in a hybrid topology [2]. Unlike previous RDCN designs, Duo supports multi-hop, integrated routing in a work-conserving (in a store-and-forward sense) manner. That is, unlike RotorNet [48] or Sirius [53], it does not require specialized queues or flow scheduling but can be implemented using regular store-and-forward switches. At its core, Duo builds on the structural properties of de Bruijn graphs, which allow using standard IP routing and longest prefix matching with small forwarding tables and standard transport layer protocols such as NDP or TCP. In addition to the system design, we present centralized and distributed algorithms for scheduling the reconfigurable links. We implement Duo in a packet-level simulator and empirically evaluate it under different traffic scenarios. In addition to showing its superiority in throughput over state-of-the-art static and dynamic networks, the evaluation assesses its sensitivity to parameters and analyzes its routing behavior. Finally, the thesis demonstrates the feasibility of Duo with a proof-of-concept implementation of the control and data plane with standard programmable network stacks.

The fifth and third major contribution brings macroscopic reconfiguration-awareness to RDCNs. Therefore, this thesis integrates both reconfiguration classes in a single RDCN design. The resulting architecture combines static, *DO* (*rotor*), and *DA* switches into a hybrid topology. This thesis explores two specific system proposals for such a system: Cerberus [3] and Trio [4].

[1]For sake of readability, we use only the term CDN when referring to the end-user mapping. It implicitly includes also CP. Chapter 5 provides a more detailed description of what networks are considered.

Building on the observation that the best-suited reconfiguration class depends on the type of traffic or, more generally, the traffic pattern, CERBERUS matches the traffic with the best topology reconfiguration class. Specifically, it forwards the traffic to the respective topology part. This provides reconfiguration-awareness to the flows on a macroscopic level, i.e., it only considers the reconfiguration class but not individual reconfigurations. For instance, latency-sensitive flows are forwarded via the static topology part; large flows use direct connections as provided by *DO* switches, and medium-sized all-to-all traffic is sent via *rotor* switches. Using flow-level simulations, this thesis shows that CERBERUS outperforms other RDCNs regarding throughput.

TRIO re-uses the general idea but elaborates on an end-to-end system design in more detail. First, it leverages the de Bruijn-graph-based base topology from DUO. Moreover, it presents a data and control plane design that allows reconfiguring individual links, according to the reconfiguration class, and also adjusting the sizes of the sub-topologies. That is, it can convert *DA* to *DO* switches during run-time and vice versa. The packet-level simulation-based evaluation illustrates that TRIO increases the goodput compared to DUO and other state-of-the-art solutions. Moreover, it maintains competitive flow completion times and shows slight improvements in resource efficiency.

1.3 Thesis Organization

This thesis is organized as follows.

Chapter 2 presents a more in-depth motivation for *DA* networks. Therefore, it provides traffic analysis for DML workloads, which serve as an example of an emerging application in DCN. In particular, it measures and evaluates the traffic originating from a selection of state-of-the-art and industry-standard DML frameworks and varying ML models and configurations.

Chapter 3 introduces basic terminology and technological background for reconfigurable optical networks for the two considered use cases, WAN and DCN. It further provides an overview of the state-of-the-art in both scenarios and introduces the top-of-rack (ToR)-matching-ToR model that can describe existing spine-leaf reconfigurable topologies.

Chapter 4 revolves around the measurement of programmable COTS equipment under reconfiguration. Therefore, it first summarizes existing measurements of equipment for topology reconfigurability. It then presents the measurement framework and describes the outcomes of the measurement campaign by extracting the dominating factors of influence on the reconfiguration delay. Finally, the chapter motivates and presents the emulation framework EXREC for hybrid topologies consisting of *DO* and *DA* components. The emulation framework is verified and used to illustrate the benefits of such hybrid topologies.

Chapter 5 addresses network operation and optimization with reconfigurations on multiple layers in the WAN scenario. First, it describes the challenges that arise with the dominance of CP in ISP networks and presents the chapter opportunities. Afterward, it formalizes the joint optimization framework and presents two solution methods. Finally, it evaluates the proposed approach using real data from a large European ISP and discusses potential deployment options.

Chapter 6 concerns with the design of reconfigurable, demand-aware DCNs. To motivate DUO, the chapter qualitatively compares existing RDCNs designs with respect to routing approaches. It then presents DUO, a high throughput RDCN, that features local routing and control decisions. DUO is evaluated using packet-level simulations as well as a proof-of-concept implementation.

Finally, Chapter 7 proposes a step towards reconfiguration-aware RDCNs by presenting and evaluating Cerberus and Trio. It first describes and evaluates Cerberus, a three-headed topology design that matches flows with the best-suited reconfigurable topology. Afterward, it introduces Trio which extends the basic structure and insights from Duo (Chapter 6) with the reconfiguration-aware design from Cerberus. The chapter describes important data and control components and mechanisms and evaluates Trio using packet-level simulations.

Chapter 8 concludes this thesis and discusses possible directions for future work.

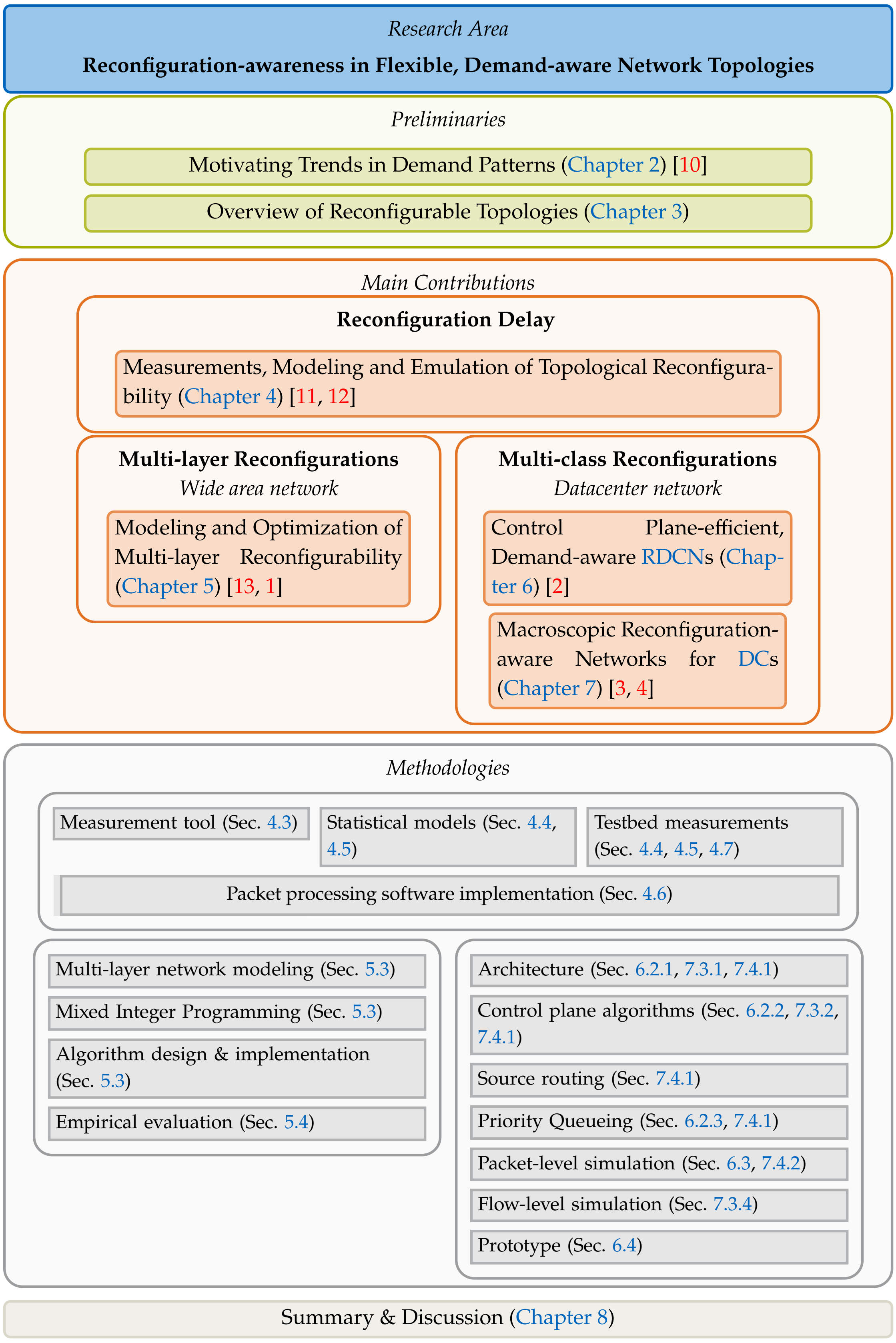

Research Area
Reconfiguration-awareness in Flexible, Demand-aware Network Topologies
Preliminaries
Motivating Trends in Demand Patterns (Chapter 2) [10]
Overview of Reconfigurable Topologies (Chapter 3)
Main Contributions
Reconfiguration Delay
Measurements, Modeling and Emulation of Topological Reconfigurability (Chapter 4) [11, 12]
Multi-layer Reconfigurations
Wide area network
Modeling and Optimization of Multi-layer Reconfigurability (Chapter 5) [13, 1]
Multi-class Reconfigurations
Datacenter network
Control Plane-efficient, Demand-aware RDCNs (Chapter 6) [2]
Macroscopic Reconfiguration-aware Networks for DCs (Chapter 7) [3, 4]
Methodologies
Measurement tool (Sec. 4.3)
Statistical models (Sec. 4.4, 4.5)
Testbed measurements (Sec. 4.4, 4.5, 4.7)
Packet processing software implementation (Sec. 4.6)
Multi-layer network modeling (Sec. 5.3)
Mixed Integer Programming (Sec. 5.3)
Algorithm design & implementation (Sec. 5.3)
Empirical evaluation (Sec. 5.4)
Architecture (Sec. 6.2.1, 7.3.1, 7.4.1)
Control plane algorithms (Sec. 6.2.2, 7.3.2, 7.4.1)
Source routing (Sec. 7.4.1)
Priority Queueing (Sec. 6.2.3, 7.4.1)
Packet-level simulation (Sec. 6.3, 7.4.2)
Flow-level simulation (Sec. 7.3.4)
Prototype (Sec. 6.4)
Summary & Discussion (Chapter 8)

Figure 1.1 Thesis structure. The main contributions belong to three fields: measurements of topology reconfigurations and design of evaluation frameworks, networks with reconfigurations on multiple layers, and networks with multiple classes of topology reconfigurations. Whereas the first field focuses on understanding the impacts of topological reconfigurations with hardware measurements and testbed implementations, the second and third present system designs and resource management algorithms. These are evaluated using simulations (both) and prototype system implementations (only third aspect).

Chapter 2

Motivating Trends in Network Demand Patterns

This chapter illustrates the traffic characteristics of a high-bandwidth application in datacenter networks (DCNs). Thereby, it shall motivate the use of reconfigurable datacenter networks (RDCNs) for such applications and sketch the opportunities that arise with the observed traffic patterns. The investigated application is machine learning (ML) which is being applied to more and more use cases ranging from business to health to entertainment [67, 68, 69].

This increased demand for high-performance models comes with an increase in computational demand which, in turn, leads to the distribution of the workload across multiple machines [70, 71]. With distributed workloads, the computational restraints shift to communication bottlenecks [72], and a variety of frameworks for distributed training of ML models (DML) has been proposed. These frameworks take communication into account [73, 74, 75, 76, 77, 78], e.g., by modifying the communication structure. However, a detailed evaluation and comparison of the communication structure of such frameworks is missing.

This chapter overviews and characterizes the communication behavior of three state-of-the-art, industry-standard distributed machine learning (DML) frameworks. It analyzes traffic traces from running these DML frameworks in more than 75 scenarios varying the trained models, the framework configurations, and the packet loss in the network. Further, the chapter provides insights into the flow patterns and the network's throughput. The study reveals that traffic can largely vary across the frameworks. While some frameworks exhibit well-predictable patterns, a desired property for demand-aware (*DA*) networks, others are less structured. Another cost aspect of DML training is presented by relating networking resource consumption to ML training metrics such as accuracy. The results can inform the networking community about traffic characteristics and contribute toward generating realistic DML traffic for simulation studies.

Content and Outline: Section 2.1 briefly overviews DML and the available frameworks. Section 2.2 lists related traces or generation methodologies for DML traffic. Section 2.3 describes the testbed and measurement procedure. Section 2.4 provides the flow-level analysis of the communication behavior and outlines potential influence factors of bandwidth consumption. Section 2.5 summarizes this chapter and discusses possible avenues for future work.

This chapter is entirely based on the work of a research internship that was supervised by me. The results were presented in a previous conference publication [10]. As a contribution to the research community, the code (`https://github.com/tum-lkn/dml-network-traffic-analysis`) and the collected traces have been made publicly available [27].

2.1 Background

This section gives a short introduction to ML and DML. It further overviews commonly used communication structures and libraries for DML and finally, presents available DML frameworks.

2.1.1 Machine Learning Training

Machine learning can be defined as the iterative usage of the training dataset to gradually fit a model, a Neural Network (NN). Each repetition in this iterative process applies a function to the dataset for the successful determination of the NN's parameters. The backpropagation algorithm is one of the most popular algorithms for training an NN. As introduced by Rumelhart, Hinton, and Williams in 1986, backpropagation is known as an efficient way to compute the gradient of the cost function with respect to the NN parameters [79]. Backpropagation is utilized as the standard approach in NN training.

Backpropagation calculates weight updates proportional to the error propagated back from the output through the different layers of the NN. These weight updates are generally done in batches, i.e., chunks of the training dataset. The training consists of numerous iterations until the prediction error drops below a threshold or a maximum of iterations is reached.

The numerous iterations are computationally intensive, in particular as dataset and model sizes grow. However, the backpropagation algorithm can be formulated as a matrix-vector multiplication operation, and thus, it can be parallelized across different workers using MapReduce [80] approach. The map stage computes the per-worker update of the NN, and the reduce stage gathers and aggregates these updates. The following sub-section describes different approaches of this distribution in more detail.

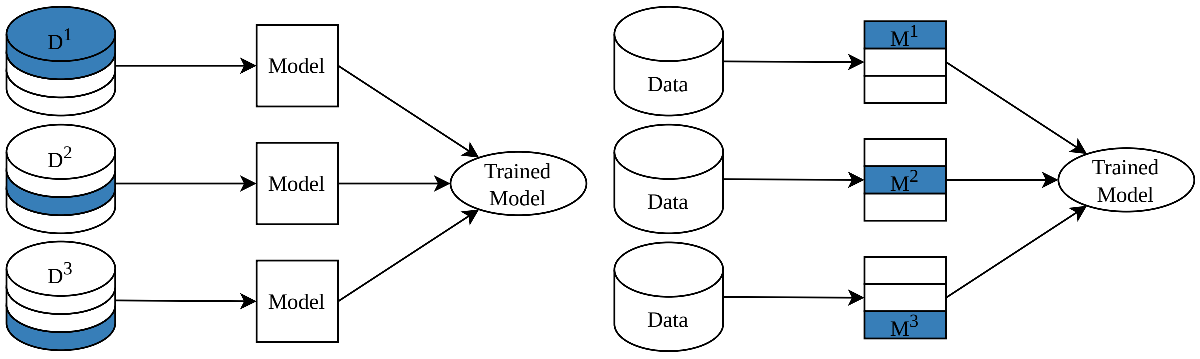
2.1.2 Distributed Training

Distributed training splits the total workload among multiple workers in order to parallelize and speed up the overall training process. In general, there are two fundamental flavors of distributed training:

Data Parallelism. Data parallel methods split and distribute the dataset across the available workers (Figure 2.1a). Each worker holds a local copy of the entire model and performs the update steps on the local copy. All workers synchronize their copy of the model by exchanging their gradient information, e.g., using the parallelized variant of the Stochastic Gradient Descent (SGD) [81]. This step is referred to as *all-reduce*.

This process can be done either *synchronously* or *asynchronously*. In the synchronous case, each worker waits until all the other workers complete their calculations and only then exchanges parameters. In the asynchronous case, each worker runs at its own pace without being affected by the slow workers. This approach increases the fault tolerance, as the training job will continue to function even if one of the nodes fails. On the downside, it introduces stale gradients, in which the slow workers train and update an old model version [82].

Model Parallelism. Model parallelism partitions the model and distributes the parts across the available workers (Figure 2.1b). It is beneficial if the model's size exceeds a single node's resources, e.g., the memory of the graphics processing unit (GPU). Since the layers of the model



(a) Data Parallelism. **(b)** Model Parallelism.

Figure 2.1 Two approaches to parallelization of distributed training: Data and Model Parallelism. Figure based on [84, Fig. 2].

are evaluated sequentially, a naïve implementation leads to underutilization of the available computing resources. Pipeline parallelism has been introduced to mitigate this. It subdivides a batch of data into micro-batches which are evaluated on the different model parts in a pipelined fashion [83].

Since most commonly used models still fit onto single nodes, model parallelism is less often applied in practice. Therefore, this chapter focuses on data parallelism.

2.1.3 Communication Topologies

The communication topology describes how the workers logically interconnect and how updates are relayed and aggregated across the set of workers. There are several popular communication structures such as trees, rings, and parameter server (PS) architectures, which are being used in practice for DML clusters [84]. The synchronous all-reduce paradigm is used with mesh, tree, and ring structures, whereas asynchronous training is generally implemented on PS architectures [85].

During the all-reduce phase, each worker sends its model updates to all other workers. When using a ring structure, a worker sends its updates only to one other worker that forwards the updates to the next worker and so on, forming a ring-like structure. Model updates received from other workers are forwarded along the ring too until the message has reached all workers. For tree structures, the communication and the exchange of updates are relayed via the respective parent nodes where one worker is the root. PS uses a more centralized approach: All the variables are stored in one or multiple PSs and workers communicate with the PSs for pushing and pulling updated gradients. One worker is designated as the *chief* and coordinates the training process. The various communication topologies have different requirements and hence multiple communication libraries exist in the literature [86].

2.1.4 Communication Libraries

The two main communication primitives are point-to-point communication and collective communication. Point-to-point communication serves the purpose of transmitting a message between a pair of processes, whereas collective communication is used for the transmission of messages among groups of processes. Frequently used libraries that implement these communication primitives include Google Remote Procedure Call (gRPC) [87], Message Passing Interface (MPI) [88], Facebook's Gloo [89] and NVIDIA Collective Communication Library (NCCL) [90].

gRPC relies on point-to-point communication, in the spirit of client/server communication where a client can directly call a method on a server. The library has no collective communication support.

MPI is the most often used, de facto standard for high-performance computing and supports collective communication [86]. It has many implementations ranging from the widely used open-source OpenMPI library to commercial ones.

Gloo is a collective communications library developed by Facebook specially for ML applications. It provides implementations for broadcast and all-reduce.

NCCL is developed to achieve high bandwidth over PCIe and NVLink between GPUs in a single node, or over Ethernet and InfiniBand for connections across machines. It is optimized for NVIDIA GPUs, implementing multi-node and multi-GPU communication standards supporting collective communication as well as point-to-point communication. It is often used for accelerating collective communication in DML [74].

2.1.5 Distributed Machine Learning Frameworks

DML frameworks build around the presented communication libraries and topologies and basic ML libraries such as `TensorFlow` [75] or PyTorch [77], and orchestrate the distributed training process.

The DML survey in [84] presents a thorough overview of the existing libraries and frameworks for distributing the training workload. The evaluation in this chapter focuses on three frameworks which provide *Data Parallelism*, and represent the state-of-the-art in research and/or are adopted by industry: `TensorFlow` Distributed [75], `Horovod` [73] and `KungFu` [74].

TensorFlow Distributed is the distributed training implementation provided by `TensorFlow`. It can distribute tasks across multiple GPUs, tensor processing units (TPUs), or machines employing data parallel methods. `TensorFlow` Distributed uses gRPC as the communication library and supports synchronous and asynchronous communication strategies through ring all-reduce algorithms and parameter server strategies on top of gRPC, and NCCL's all-reduce algorithms [86].

Horovod is an industry-standard distributed deep learning training framework supporting `TensorFlow`, Keras [91], PyTorch [77], and Apache MXNet [92] as ML libraries. It distributes the training process with minimal code addition to the single-node training scripts and supports only synchronous communication strategies through ring all-reduce algorithms. `Horovod` can use Gloo or MPI for collective communication. It also supports NCCL for tensor operations, however, primitive administrative operations such as gathering the number of workers and the ranks of the workers require MPI or Gloo.

KungFu is a framework for adaptively distributing the ML workload. It provides synchronous and asynchronous training and includes an online monitoring component that adapts the communication architecture of the workers according to the network state during training. Its distributed synchronous optimizer (referred to as *Synchronous SGD* optimizer) is equivalent to the one in `Horovod`. The asynchronous optimizer (referred to as *Pair Averaging* optimizer) is the implementation of Asynchronous Decentralized Parallel SGD [93]. `KungFu` includes its own collective communication API which can be accelerated via NCCL. Moreover, `KungFu` provides a variety of topologies for the communication patterns of the workers. These include Tree, Star, Clique, and Binary Tree Star (BTS).

2.2 Related Work

There are several relevant works, which shed light on the characteristics of network traffic of different DML frameworks from different perspectives. From a computing resources perspective, Jeon et al. provide insights about jobs' requirements and analysis on GPU usage from large-scale DML production clusters [94]. But they neglect the communication aspect and do not analyze the network traffic in detail.

Since several studies have shown that communication is a bottleneck to distributed training, like [72], many research papers focus on improving the communication efficiency such that increasing the number of GPUs linearly scales out the performance. Such solutions include, among others, compressing the gradients [95], coordinating the distributed gradient computations to micro-manage the communication patterns of nodes [96], pruning model parameters [70] or scheduling communication [97]. They use additional (local) computation to reduce communication and focus on improving the total training time, but they leave aside implications on the network traffic and analysis thereof.

Existing literature has also considered the networking perspective to achieve linear scale-out. These works propose to modify the all-reduce architecture to hierarchical all-reduce [98], network-based systems such as in-network aggregation to overcome communication bottlenecks [99, 100], flow schedulers [101], or tailored topologies [102, 103, 104]. They contain descriptions of DML network traffic but focus on point solutions for specific frameworks, their configurations, or communication patterns. For instance, Liu et al. [103] assume only PS-based DML, whereas Gebara et al. [100] consider only ring-reduce DML traffic in their evaluation. This chapter provides analyses and compares traffic patterns as observed from multiple publicly available frameworks and varying configurations.

In addition to proposing new designs, some works examine different approaches' training throughput in terms of image per second trained and their scalability [105, 106]. Other works aim to measure the network performance of distributed training approaches as in [72, 86]. All these researches focus on identifying the communication bottleneck to the total training time but leave detailed network traffic patterns aside. There exists prior works which assesses the performance of DML on various topologies [100, 107]. However, these works include simplified assumptions about flow distributions and the used DML frameworks. That is, they consider given flow sizes and only homogeneous communication structures, e.g., only PS. Finally, there are papers outlining link utilization metrics of DML training [108]. However, these papers lack either distinction between frameworks and model sizes, or do not specify temporal patterns.

The work closest to this study is Driple [109] which uses a Graph neural network to predict the resource consumption of DML training jobs. However, Driple focuses on the prediction of burst duration and amplitude of training jobs whereas this study is concerned with the overall traffic patterns in the network. Overall there is no comprehensive analysis of the network traffic of different DML frameworks and their configurations.

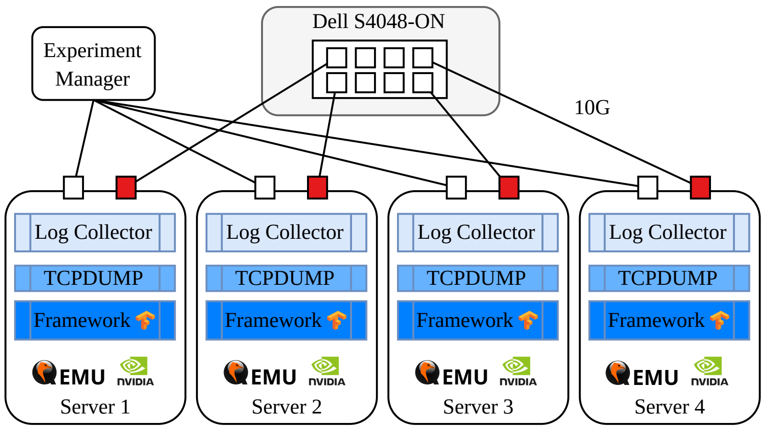


Figure 2.2 Overview of the testbed to collect network traffic from DML frameworks. Four servers are connected to a 10Gbps switch. Each server has one GPU and runs a virtual machine (VM) with a worker of the DML framework. Traffic traces and application logs are collected inside the VMs. An *Experiment Manager* orchestrates the measurement.

2.3 Measurement Setup

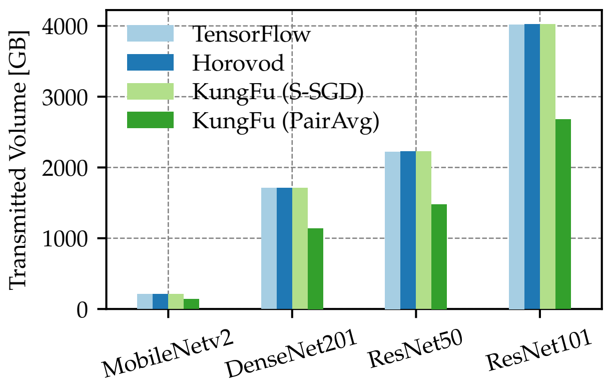
The goal of this chapter is to explore and compare the network behavior of frameworks and models on a small representative setup. This section describes the testbed and the measurement procedure for tracing the network traffic.

2.3.1 Testbed

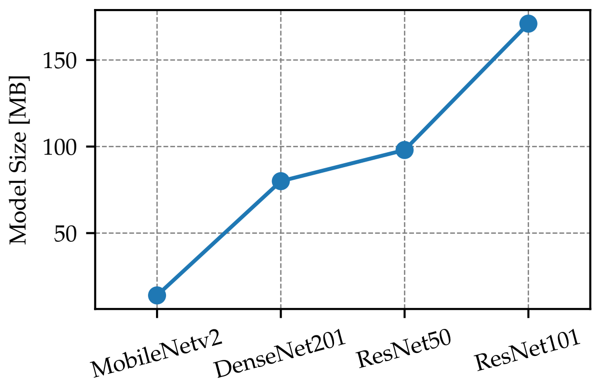
Figure 2.2 shows the testbed. It consists of four servers running Ubuntu 18.04 (5.15.0-47-generic kernel) with 128 GB of RAM and Intel Xeon Silver 4114 @ 2.2 GHz (20 cores). Each of the servers contains one NVIDIA Tesla T4 GPU and is connected via a 10 G Ethernet port (red box) to a Dell S4048-ON switch in L2 forwarding mode. The servers run kernel-based VMs with 48 GB of RAM and 8 pinned central processing unit (CPU) cores. Each of the VMs is running the latest version of "generic/ubuntu1804" Vagrant Box. The GPU and network interface card (NIC) are handed over to the VMs using PCI pass-through capabilities. Communication libraries and frameworks are installed using the default instructions from the respective web pages or repositories. Specifically, this study uses `TensorFlow` 2.3.0, `Horovod` v0.22, `KungFu` 0.2. Each VM is configured to collect packets originating from itself via *tcpdump*. Moreover, application-level logs are collected with respect to each training step (training accuracy, loss, and step duration) for further analysis. A fifth server, the *Experiment Manager*, connects via a dedicated network and orchestrates the measurement.

2.3.2 Settings

In order to analyze and understand the structure of the communication pattern, four models of different sizes are trained: MobileNetv2 (14 MB) [110], DenseNet201 (80 MB) [111], ResNet50 (97 MB) [112] and ResNet101 (171 MB) [112]. The models are available in the most popular deep learning libraries such as `TensorFlow` and are used as benchmarks in similar studies [86, 74]. The training uses the CIFAR10 [113] dataset. It consists of 60 000 32x32 color images with 50 000 training and 10 000 test samples. If not stated otherwise, the batch size per worker is 64. All



(a) Total transmitted volume.



(b) Model size.

Figure 2.3 Comparison of total transmitted volume for 20 epochs of DML training across four models and four framework configurations (a). Traffic volumes correlate with the size of the model being trained (b). The total transmitted volume correlates to the model size.

models are initialized with random weights (cold start). Moreover, the study uses the Adam optimizer [114] and trains for 20 epochs where an epoch considers all samples in the training dataset. The evaluation covers the following configurations of the frameworks:

- `TensorFlow` Distributed Framework: Ring-Reduce Synchronous SGD (S-SGD) on GPU;
- `TensorFlow` Distributed Framework: Parameter Server Training on CPU (PS);
- `Horovod` Framework: Ring Reduce and Hierarchical Reduce S-SGD on GPU;
- `KungFu` Framework: S-SGD on GPU;
- `KungFu` Framework: Asynchronous Decentralized Parallel SGD on GPU (PairAvg).

For further variation, the `Horovod` configuration is run with NCCL, MPI, and Gloo as communication libraries. Similarly, `TensorFlow` is evaluated with gRPC and NCCL. For `KungFu`, BTS, Tree, Star, and Clique connectivity patterns are run. The study excludes additional optimization strategies described in Section 2.2 since they are not an inherent component of the frameworks. All the measurements involve four nodes (workers), except PS training, which includes six nodes (four workers, one PS, and one chief). Each of the framework settings and models is measured once – more than 75 scenarios in total.

2.4 Evaluation

This section presents the observations from the packet traces collected during the distributed training process in order to compare the network traffic of different frameworks, models, and communication backends. First, the section analyzes the volume of data transmission and elaborates on the connectivity patterns of different setups. Then, it conducts a flow-level analysis of the network architecture and assesses the impact of the network's condition. Finally, the achieved training accuracy is related to the transmitted network traffic.

2.4.1 Total Data Transferred

Figure 2.3 shows the total transmitted volume per model and framework together with the model sizes. The volumes for 20 epochs range between ≈ 140 GB and ≈ 4 000 GB. The trained model is the dominant influencing factor on the total transmitted volume. `TensorFlow`, `Horovod` and `KungFu` S-SGD distribute the SGD with the same approach. Hence, they transmit the same total amount of data. Compared to these, `KungFu` PairAvg is more communication efficient; it transmits ≈ 30% fewer data for all the models. For instance, it transmits 2677 GB for ResNet101, whereas the other frameworks transmit ≈ 4020 GB. Other influencing factors in the frameworks' configuration are not observed: Repeated measurements with RMSProp [115] instead of Adam, or NCCL instead of gRPC or MPI, do not differ in the transmitted volumes.

Figure 2.3b presents the sizes of the trained models. The model size is positively correlated with the total amount of data transferred during a run. This is in line with similar assessments [100]. Larger models have more variables. Thus, the gradients that are exchanged in the reduce stage, are expected to be larger, and therefore, more data is transmitted.

2.4.2 Communication Patterns

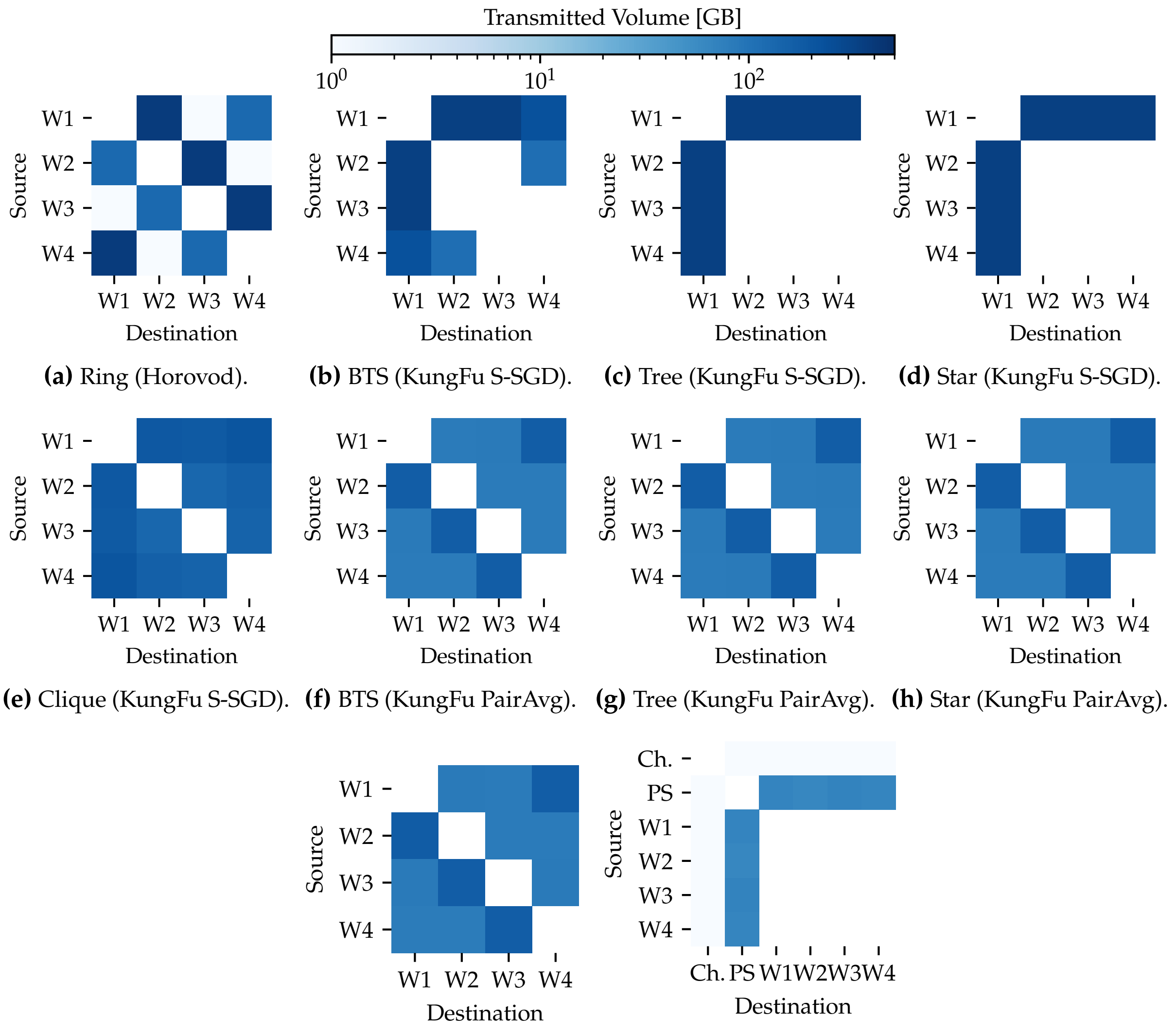
Figure 2.4 shows heatmaps of the total transmitted volume between nodes to illustrate the communication pattern of various frameworks and configurations. The darker the cell, the more data has been transmitted between a pair of nodes.

For the ring-reduce pattern (Figure 2.4a), the dark squares indicate the neighboring nodes in the ring. In this case, the structure of the ring is $W1 - W2 - W3 - W4 - W1$. In all frameworks, the user can specify the order of the ring. On average, the intense communication pairs in the ring accumulate 566 GB, whereas the opposite direction in the ring amounts to less than 1 GB. These opposite flows are constituted from acknowledgments. Communication happens only along the ring. The "non-ring" elements in the matrix are three orders of magnitude smaller. `KungFu` S-SGD with BTS (Figure 2.4b) illustrates a pattern where W1 is the root of a tree and relays all the communication between the other nodes. W4 is a child of W2 and relays 1/3 of the traffic through it. The reason for the direct communication between W1 and W4 could not be clarified. Tree and Star communication topologies use only W1 as relaying node (Figure 2.4c & 2.4d). The Clique structure (Figure 2.4e) shows the densest pattern for S-SGD with data transmissions for all communication pairs. `KungFu` PairAvg (Figure 2.4f-2.4i) consistently employs a full-mesh pattern. A more intense structure similar to the ring pattern is evident. Finally, the PS configuration (Figure 2.4j) shows that workers are communicating only with the parameter server. No cross-communication among the workers is observed. The chief distributes the tasks to all the workers and the parameter server.

In conclusion, the framework and its configuration can have a strong impact on the patterns of network traffic. In particular, Ring, tree, and PS structures exploit predictable communication patterns with high volumes between few pairs are evident. A repetition of this evaluation with eight workers confirmed that the observations are also valid for larger scenarios.

2.4.3 Flow Analysis

Having analyzed the total traffic, we are now interested in how many network flows are present during training and how the traffic is distributed among the flows. A flow is defined as the



(a) Ring (Horovod). **(b)** BTS (KungFu S-SGD). **(c)** Tree (KungFu S-SGD). **(d)** Star (KungFu S-SGD).

(e) Clique (KungFu S-SGD). **(f)** BTS (KungFu PairAvg). **(g)** Tree (KungFu PairAvg). **(h)** Star (KungFu PairAvg).

(i) Clique (KungFu PairAvg). **(j)** PS (TensorFlow).

Figure 2.4 Comparison of DML communication patterns: heatmap of total transmitted volume per DML framework configuration, averaged over all models. "Ch." indicates the *Chief* node in Figure 2.4j Connectivity patterns depend on the distributed optimizer and training strategy in use.

five-tuple of source and destination IP addresses, transport protocol, and source and destination ports.

2.4.3.1 Flow Structure

Figure 2.5 shows bars indicating the total number of flows per framework and model. The parts of the bars (indicated by the hatches) correspond to different flow sizes. The lower part is the number of small flows (< 6 GB) and the top part are flows with a size ≥ 6 GB. The threshold is chosen such that the accumulated volume of all large flows is larger than 95% of the total transmitted volume. The total number of flows in a framework is constant across models except for `Horovod` and `KungFu` with the Clique topology. Moreover, the total number of flows in a distributed training scenario depends mainly on the number of workers and the communication strategies. Repeated measurements with the same settings indicate that the number of flows does not change across runs with the same configuration. The number of flows ≥ 6 GB varies across

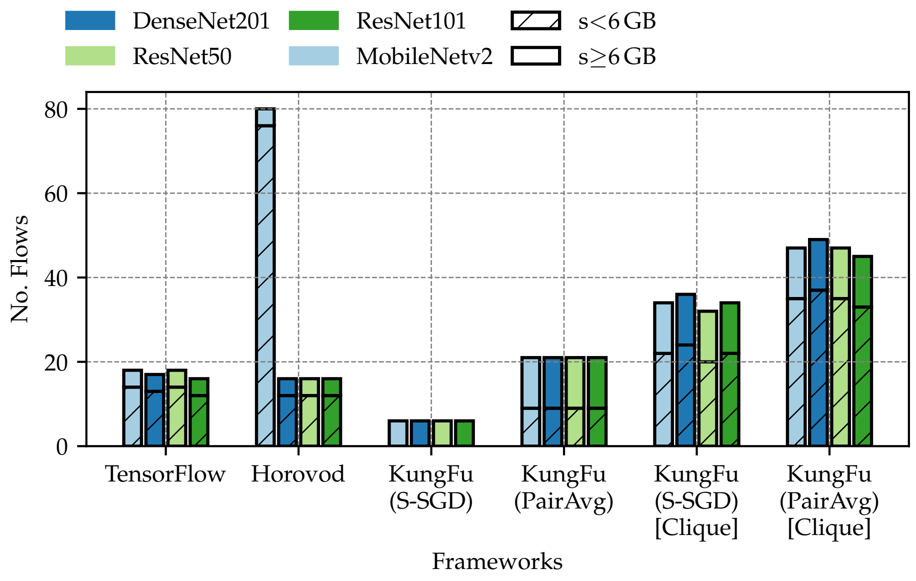


Figure 2.5 Comparison of number of flows per DML framework and model. The stacked bars (indicated by the hatch) separate flows by their size *s* into small and large ones (≥ 6GB). `TensorFlow` and `Horovod` represent ring topology. For `KungFu`, BTS, Tree and Star topologies have the same number of flows, whereas Clique topology has more flows.

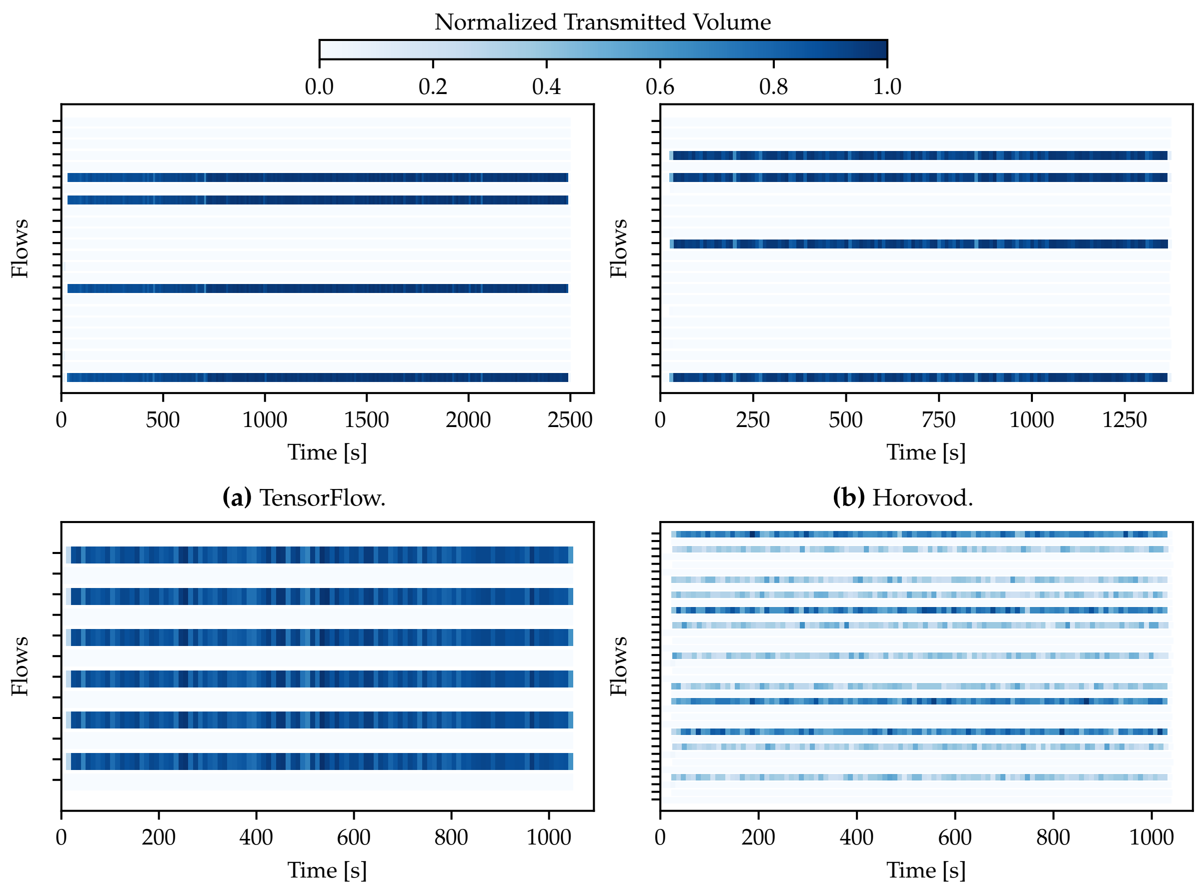
the configurations. For synchronous training, there are four big flows in the ring topology, six big flows in BTS and Tree, and eight big flows for the case of the asynchronous PairAvg optimizer. While the chosen threshold ensures that the large flows make at least 95% of the total transmitted volume, we observe that they even can contribute up to 99% on average (not shown in the figure). Overall, we observe more variance across models and frameworks for small-sized flows.

`TensorFlow` with ring all-reduce strategy has 24 flows. For four workers in the ring structure, four main flows exchange gradients (large flows), and another four flows serve the acknowledgments in the reverse direction (part of the small flows). However, `TensorFlow` opens side connections between the non-neighboring nodes in the ring. The traces contain 16 of such flows which each amounts only to 1 MB traffic on average with 912 KB variance.

`Horovod` has 24 flows except for a single MobileNetv2 measurement with 88 flows. When the VM and `Horovod` are set up for the first time, `Horovod` runs initial checks between the servers and this causes the number of flows to grow to 88. This behavior is repeated every hour due to caching mechanisms inside the framework. Although the number of flows in `TensorFlow` and `Horovod` is the same, the latter one uses eight flows for SSH connections.

`KungFu` S-SGD, using BTS, Tree and Star communication topologies, has 12 flows in total, consisting of 2 flows in both directions for each edge of the tree. The Clique topology serves 46 flows on average. We observe that only the number of small increases (lower part of the bar) while the number of large flows is the same as for the other communication topologies with `KungFu` S-SGD. In addition to the 12 flows for serving gradient exchanges, it has 22 small flows consisting of two sizes with 54 Bytes and 74 Bytes. The remaining flows have an average size of 63 MB.

`KungFu` PairAvg creates the most flows. Its pattern has 36 flows in total for BTS, Tree and Star communication topologies and 64 for Clique topology. In both cases, the number of flows for both classes increases. Focusing on BTS, W1 has four flows with the other workers in both directions making up 24 flows (cf. Figure 2.4f). The remaining 12 flows are between the rest of



(a) TensorFlow. **(b)** Horovod. **(c)** KungFu (S-SGD; BTS). **(d)** KungFu (PairAvg; BTS).

Figure 2.6 Throughput per flow over time. Horizontal bars show individual flows. The opacity of the bar indicates the throughput per 10 s time window normalized to the maximum throughput in a scenario. The trained model is ResNet50. `KungFu` PairAvg distributes traffic over more flows than the other scenarios.

the workers in both directions. Since W1 is the root node of the tree, it has double the number of connections. Similar to the configurations with S-SGD, small flows that amount to 54 Bytes and 74 Bytes, make up the difference in flow numbers between Clique and the other communication topologies. Overall, we conclude that the chosen communication strategy impacts the number of flows independently of the framework.

2.4.3.2 Temporal Behavior of Flows

Figure 2.6 illustrates the transmission behavior of the flows over time.[1] For each flow, it illustrates the transmitted data per 10 s time window. The more opaque a cell of the bar, the more data is transmitted in that time window. The values are normalized to the maximum value per scenario.

In line with the previous observations for the number of flows, the large share of the bars is transparent, i.e., the corresponding flows do not carry a large volume. They are present during the whole run. However, the transmitted volume is low. Also, the large flows (dark bars) transmit uniformly throughout the whole training process. They serve the gradient exchanges which contribute towards the learning process.

[1]We omit the PS scenario in the following since it was trained on the CPU and hence the timings are not comparable.

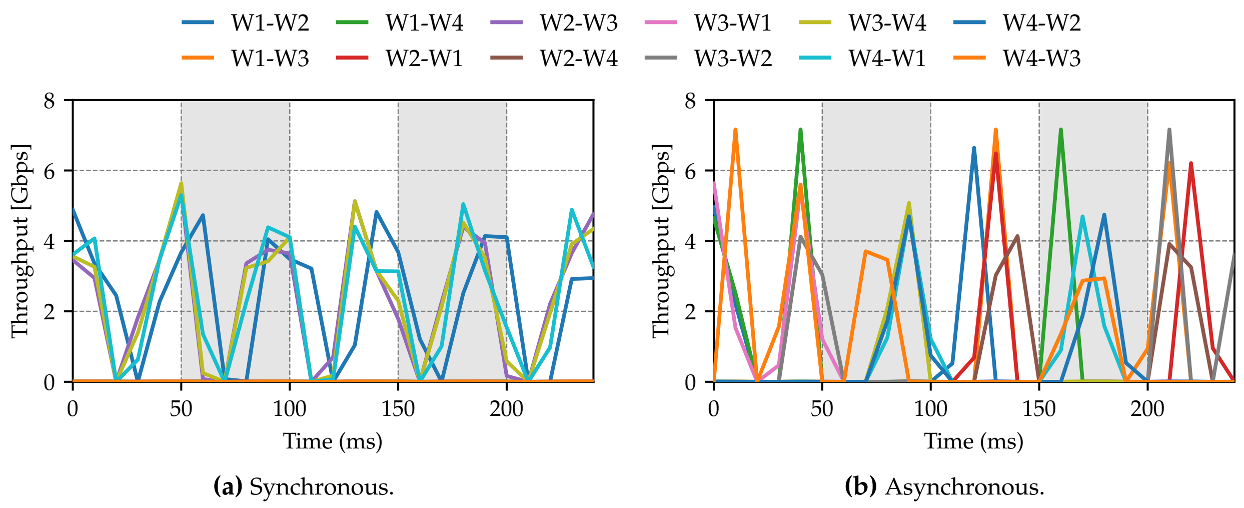


Figure 2.7 Throughput per communication pair over time. Synchronous communication shows clearly periodic behavior (a). Less structure is evident in the asynchronous case (b). The ring structure of the synchronous case has only 4 flows forming the ring; the other flows are 0.

For the ring-reduce patterns employed by `TensorFlow` (Figure 2.6a) and `Horovod` (Figure 2.6b), all the large flows start at the very beginning of the training. This is different for `KungFu` S-SGD optimizer. A closer look with one second precision reveals that two sets of flows start at the very beginning and the other two start slightly after. The BTS pattern causes this difference. All the traffic is relayed via the root node of the tree. There is a slight delay between receiving and sending at the root and hence the arrival times of the flows are slightly different.

In the asynchronous case implemented by `KungFu` PairAvg, flows arrive independently of each other. Besides, the data transmission shows less of a continuous pattern. This is not very much clear since all workers have the same specifications and do not deviate in terms of computation power.

The flow-level analysis reveals different patterns across the frameworks. Besides the number of flows, the intra-flow behavior differs. To assess this more in detail, the following subsection evaluates the data transmission rates of the communication pairs.

2.4.4 Intra-flow Patterns

As a first evaluation of the intra-flow behavior, Table 2.1 reports the average throughput values for the four 10 G links in the testbed. The results build on 10 measurement runs per framework configuration and additionally, each run is split into three windows to calculate the average throughput (batch mean). This leads to 30 samples in total. Note that computation phases are also included when computing the throughput and that NCCL acceleration is not used. The bold fontface indicates largest value per row.

Overall, none of the frameworks makes efficient usage of the available bandwidth. For all models, `Horovod` and `KungFu` S-SGD consistently obtain higher throughput values, followed by `KungFu` PairAvg and `TensorFlow`. Within a framework, the trained model affects the observed throughput. Smaller models, e.g., MobileNetv2 and DenseNet201, result in lower average throughput values than the larger models such as ResNet101.

Table 2.1 Average throughput summed over all links, without NCCL acceleration. Square brackets report 95% confidence intervals. The unit is Gbps.

	TensorFlow	Horovod	KungFu (S-SGD)	KungFu (PairAvg)
MobileNetv2	4.91 [4.82, 5.00]	**8.19** [7.80, 8.58]	6.96 [6.14, 7.78]	4.55 [4.06, 5.04]
DenseNet201	6.42 [6.29, 6.55]	11.59 [11.39, 11.80]	**12.33** [12.00, 12.65]	10.09 [9.88, 10.29]
ResNet50	6.81 [6.71, 6.90]	12.80 [12.12, 13.48]	**13.17** [12.68, 13.67]	11.72 [11.46, 11.98]
ResNet101	6.33 [6.19, 6.47]	**13.68** [13.37, 14.00]	13.56 [13.18, 13.95]	12.08 [11.76, 12.41]

Table 2.2 Throughput of a single flow which carries gradient exchange. Square brackets report 95% confidence intervals. The unit is Gbps.

	Average Throughput	Peak Throughput
TensorFlow - gRPC	1.65 [1.61, 1.68]	9.60
TensorFlow - gRPC - NCCL	**6.38** [6.25, 6.51]	9.78
Horovod - MPI	3.29 [3.20, 3.39]	8.07
Horovod - MPI - NCCL	**6.61** [6.55, 6.67]	9.74
Horovod - Gloo	4.70 [4.56, 4.83]	9.61
Horovod - Gloo - NCCL	**6.23** [6.11, 6.35]	9.76
KungFu (S-SGD)	2.76 [2.65, 2.87]	9.56
KungFu (PairAvg)	1.42 [1.28, 1.56]	9.62

Larger models lead to larger gradients being transmitted at the end of a step; hence, larger models lead to more data transmission. However, contrary to the expectation, the used framework can cause a bottleneck. We observe that bandwidth utilization is closely linked to the framework and communication backend in use rather than the model size. For instance, TensorFlow bottlenecks the communication at an average throughput around 4.9 Gbps for MobileNetV2 and 6.5 Gbps for the other models. However, Horovod reaches a throughput of 8.19 Gbps for MobileNetV2 exceeding TensorFlow's performance for the larger models. The lower utilization of TensorFlow is caused by the lack of collective communication support of gRPC library. OpenMPI as used by Horovod outperforms gRPC and KungFu's dedicated API further improves upon OpenMPI. Since KungFu PairAvg optimizer consumes less data overall in comparison to the other frameworks, it translates into lower bandwidth utilization.

In order to illustrate the differences between the synchronous and asynchronous training implementations, Figure 2.7 zooms into 250ms of the captured data for training the MobileNetv2 model. The figure compares the throughput of each communication pair across the two synchronization approaches. Figure 2.7a shows the synchronous case using Horovod with the ring-reduce pattern. In this case, the framework uses all connections forming the ring in a burst-and-stop fashion. The peaks correspond to gradient exchanges and the valleys correspond to computation phases. Since the training is synchronized across the workers, they exchange gradients at the same time. The asynchronous implementation uses KungFu PairAvg (Figure 2.7b). It shows peaks and bottoms at independent times without a particular structure. All the workers are pushing updates once their computation phase is complete.

Table 2.2 compares the throughput of a single flow across eight combinations of frameworks and communication backends. The flow is used for gradient exchange while training a ResNet50 model. Starting with the peak throughput, we observe that it is only little influenced by the

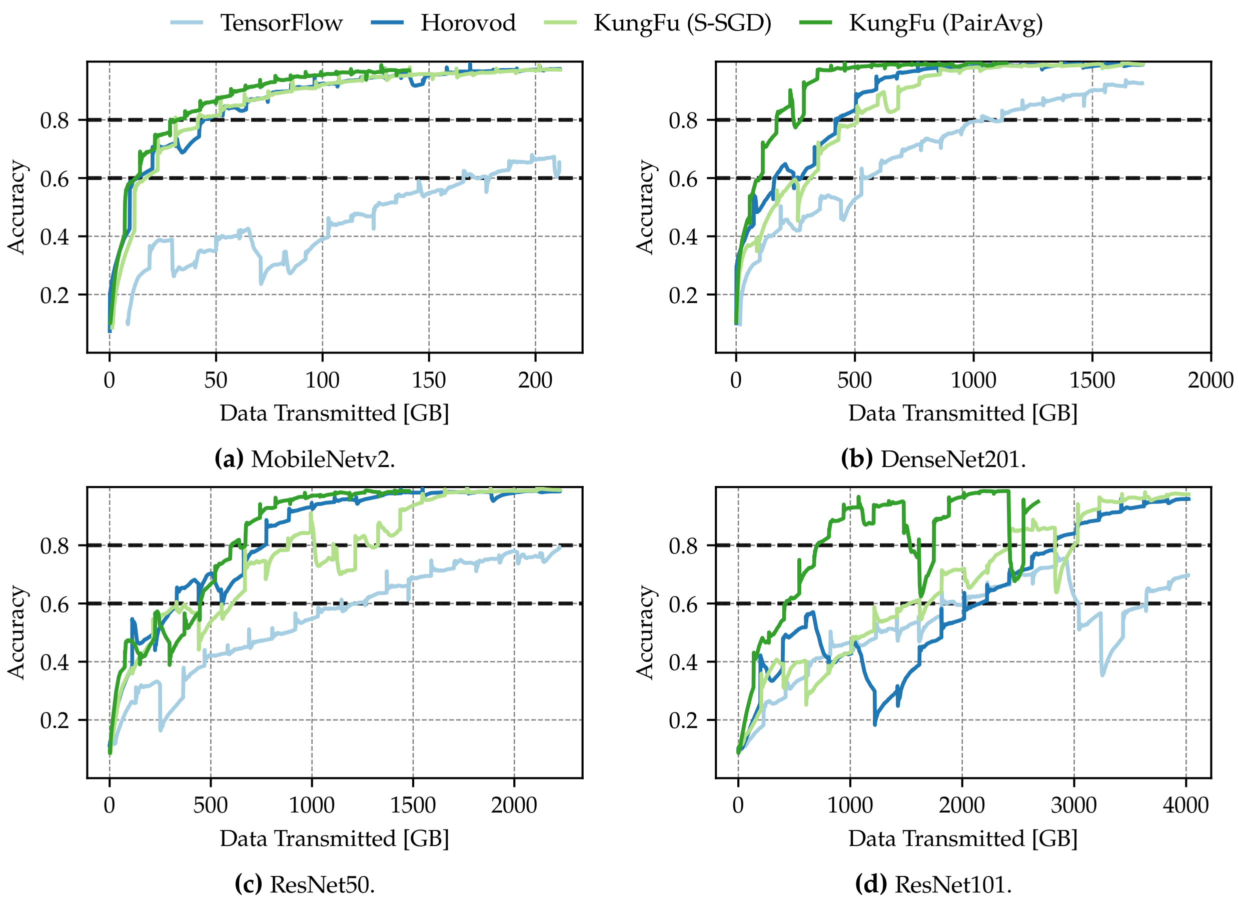


Figure 2.8 Line plot of the achieved training accuracy against the transmitted volume. Comparison of DML frameworks and models. The dashed lines indicate 60% and 80% accuracy levels respectively. Each framework exhibits diminishing gains in accuracy with respect to transmitted data.

framework and the communication backend. All values are close to the link capacity, in the range of 9.50 – 9.80 Gbps except for `Horovod` with MPI (8.07 Gbps). The behavior changes for the average throughput. The comparison demonstrates that the average throughput significantly depends on the communication backend. On average, gRPC utilizes less than 17% of the available bandwidth. MPI and Gloo make better use of the bandwidth. Without NCCL, they obtain average throughput values of 3.29 Gbps and 4.70 Gbps respectively. `KungFu` seems to underperform in comparison to other backends for a single flow. However, as it uses six or eight communication pairs depending on the training strategy, it makes up for it on the total throughput (cf. Table 2.1). Finally, accelerating gRPC, MPI, or Gloo with NCCL increases the utilization of the available bandwidth. The improved implementation of the collectives reduces the computation time and thereby, increases the average utilization to 65%.

From an intra-flow perspective, there are clear phases of computation and communication for all frameworks independently of the distribution approach. All frameworks show the periodic nature of synchronous all-reduce. However, depending on the framework, there are specific nuances in flow timings. The asynchronous case exhibits less structure.

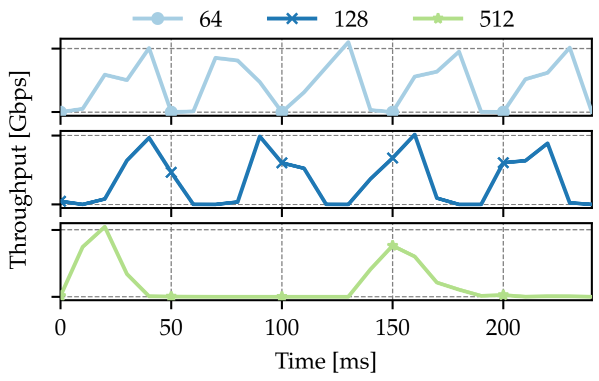


Figure 2.9 Throughput of one communication pair over time for three batch sizes. Batch size affects the periodicity of communication. The trained model is MobileNetv2.

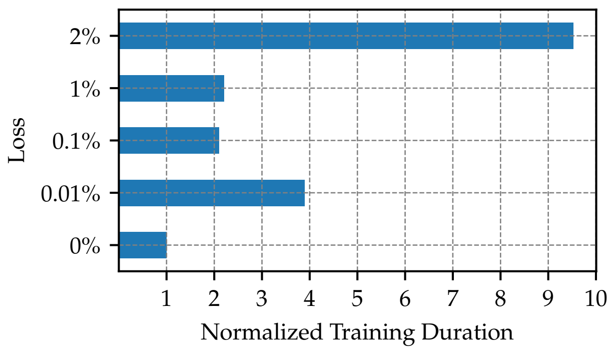


Figure 2.10 Bar plot of the training duration for different loss values. The values are normalized with respect to the measurement without loss. Introducing loss impacts the training time drastically.

2.4.5 The Cost of Accuracy

Figure 2.8 shows the accuracy on the training dataset against the accumulated, transmitted data for the considered models and frameworks. Overall, we observe the expected increase in accuracy with transmitting more data and diminishing gains in accuracy with more data transmitted. However, the exact behavior varies depending on the used framework and model. Starting with MobileNetV2 (Figure 2.8a), `TensorFlow` achieves much lower accuracy of ≈ 0.65 compared to the other frameworks which all achieve an accuracy > 0.9. This is consistent for the other models: `TensorFlow` either achieves lower accuracy or requires significantly more data to transmit to achieve comparable performance.

To compare the remaining frameworks in more detail, we evaluate the point where they first reach an accuracy of 0.8. `KungFu` and `Horovod` reach this point with < 50 GB data for MobileNetV2. Although their synchronous distribution approaches are implemented similarly (wrapping `TensorFlow`'s optimizer), their behavior starts to differ for larger models to train. For DenseNet201 (Figure 2.8b), `Horovod` needs around 410 GB and `KungFu` 510 GB; for ResNet50, the values are 770 GB and 880 GB (Figure 2.8c); and for ResNet101, they are 2.45 TB and 2.82 TB respectively (Figure 2.8d). `KungFu` PairAvg outperforms all the other frameworks as it reaches the same accuracy with around 30% less data consumption.

The used frameworks and desired training performance need to be considered when estimating the cost of training in the network. A direct relation between the amount of transmitted data and the achieved accuracy is hard to infer. The main relation is introduced by the number of training steps as elaborated by other works.

2.4.6 Impact of Batch Size

The batch size refers to the number of data samples used by each worker in a single training step. Since the DML frameworks exchange data between the workers at the end of each step, the batch size directly influences the communication-to-computation ratio. Figure 2.9 shows the throughput on a single link for batch sizes of 64, 128, and 512 samples. Bigger batch sizes increase the computation-to-communication ratio: the duration of the valleys with no communication increases. Although the batch size affects this ratio, the transferred volume per step remains the same and the communication phases on average utilize similar bandwidth to exchange the gradients. This behavior is consistent across all frameworks and models (figures are omitted for brevity).

2.4.7 Impact of Packet Loss

In order to simulate a more complex scenario with contention on links, we introduce additional packet loss on some links. Figure 2.10 relates the total training duration for MobileNetv2 trained with `TensorFlow` against the packet loss. The values are normalized by the case with 0% loss. We observe that the packet loss strongly impacts the training duration: an increased packet loss results in drastically larger training times. 0.01% loss doubles the training time; 2% loss leads to training duration more than eight times larger than for 0% loss. The performance decrease of loss-based TCP in lossy networks explains this observation [116]. Introducing loss reduces the growth of the congestion window, which results in lower throughput, longer communication phases, and, hence, increased training time.

2.5 Summary

This chapter investigates the network traffic of three state-of-the-art DML frameworks with varying configurations and training parameters. The analysis reports key network metrics such as throughput, flow, and intra-flow patterns and relates them to training accuracy. The major findings are three-fold: For all configurations, the majority of the network traffic stems from a few large flows which exist throughout the whole training process. Such a skewed pattern is well-suited for reconfigurable, *DA* networks. Further, the findings indicate that the number, and the specific spatial and temporal distribution of these large flows, and also the remaining flows, vary depending on the framework and the specific configuration. However, the transmitted volume is mainly influenced by the model that is trained.

In the context of reconfigurable, *DA* networks, this measurement campaign opens interesting future research directions and may even help improve traffic engineering and topology design for ML applications. Moreover, the insights from this chapter provide guidance for realistic traffic modeling and generation for simulation studies as well as can help users and operators to tune their datacenter (DC) architecture according to the communication needs of the training system.

Chapter 3

Preliminaries and State of the Art of Reconfigurable Optical Networks

This chapter provides general background information on reconfigurable optical networks for the two use cases considered in this thesis: wide area network (WAN) and datacenter network (DCN). The chapter first describes the considered use cases in more detail (Section 3.1) and then lists the technological enablers of reconfigurability (Section 3.2). Section 3.3 defines and illustrates major characteristics of topological reconfigurations. Finally, this chapter provides an overview of the major directions that have been explored in prior work as well as existing system designs and implementations. Following the structure of the thesis, the chapter first presents the WAN scenario (Section 3.4) followed by DCN (Section 3.5). Note that this chapter is intended to be a primer on the topics and focuses on the most important aspects only. In addition, we present a generalized topology architecture underlying the majority of the reconfigurable datacenter network (RDCN) proposals (Section 3.6). For more detailed surveys, the reader is referred to [117, 118, 119].

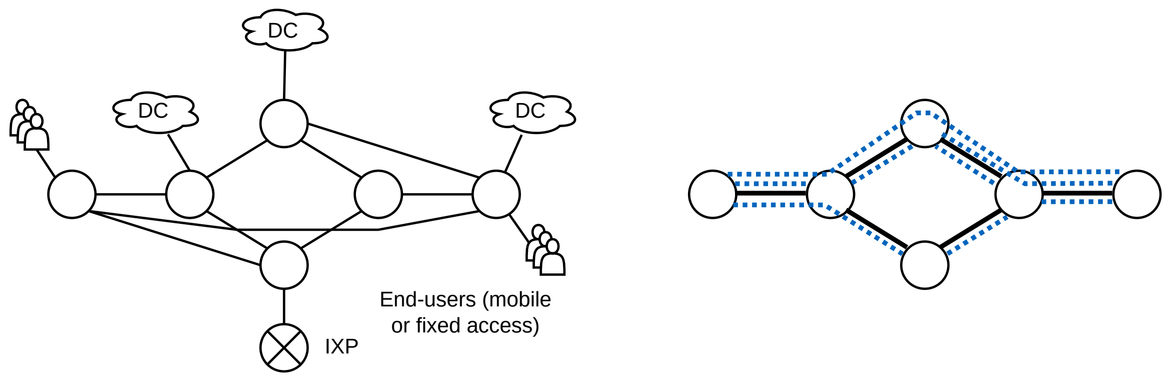
The generalized RDCN architecture (Section 3.6) is from [3] and also part of another thesis. Both authors have equally contributed to this part.

3.1 Use cases

Throughout this thesis, we will consider two use cases for reconfigurable and demand-aware topologies which are presented in the following.

3.1.1 Wide Area Networks

The WAN use case revolves around systems that implement IP-over-Optical Transport Networks (IP-over-OTN). They are mainly used to connect different metro areas or DCs and are considered in a two layer model containing the IP and the optical layer. The network endpoints can vary depending on the exact scenario. Examples are DCs, point of presences (PoPs), Internet exchange points (IXP), servers, or connections to other networks such as fixed access or mobile networks. This thesis focuses on the interconnecting, long-haul transit networks. The network endpoints connect to IP routers which in turn connect to circuit switching elements in the optical layer. The optical switching elements are connected by optical fibers. They establish and end-to-end path (using the same wavelength) between the IP routers, a so-called lightpath.



(a) IP layer.

(b) OTN layer.

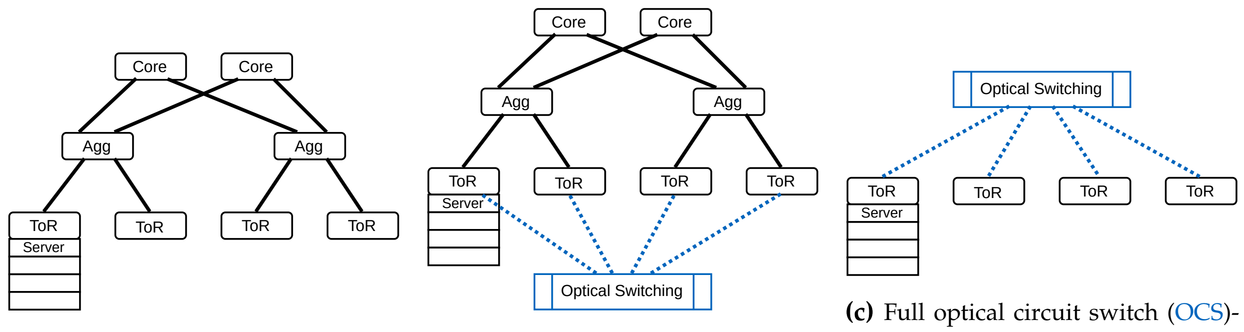
Figure 3.1 Overview of the WAN use case. Illustration of (a) IP and (b) optical layers. Circles are IP routers or optical nodes respectively. End-users, IXPs, and datacenters (DCs) connect to IP routers. Solid black lines are logical links (IP layer) and installed fiber links (OTN layer). The dotted, blue lines in (b) represent the embedding of these IP links into the OTN.

Figure 3.1 gives an overview of the scenario and the two layers. It shows a single IP-over-OTN long-haul network. In Figure 3.1a, the IP routers (circles) connect to three types of network endpoints. Fixed or mobile access networks with connected end-users are the first type. The endpoints of the second type are private network interconnects (PNI or peerings) to content providers (indicated by the DCs). These are usually created at central exchanges or PoPs of the network operators. Lastly, it connects to an IXP. Figure 3.1b shows the mapping of the IP links onto lightpaths in the OTN (blue lines). The span of the lightpaths varies. The optical switching elements can either terminate a lightpath and forward it to the local IP router, or relay it to another node.

In general, today's optical switching elements are considered reconfigurable so that the connectivity on the IP layer can be modified during the lifetime of the infrastructure. For instance, this can be used to adapt to changing or evolving demands and has already been subject to previous work, as discussed later. Since changing the underlying physical infrastructure, i.e., the installation of new fibers, occurs on a large timescale, the reconfiguration of the topology revolves around moving capacity in the IP layer. The goal is to serve all demands, or at least as many as possible, while minimizing operational expenditures (OPEX), e.g., in terms of power or deployed lightpaths.

3.1.2 Datacenter Networks

DCNs provide connectivity between a large number of end-hosts (servers) which are grouped in racks. They traditionally consist of statically connected packet switches. In order to provide high capacity between end-hosts in the network, several stages of aggregation, e.g., in fat-trees are common [42]. Figure 3.2a illustrates an example. The servers connect via the top-of-rack (ToR) switches towards the backbone. In the example, the backbone has two aggregation stages: the aggregation layer ("Agg") and the core layer. The capacity of the aggregation stages matches the capacity of the ToR uplinks. A well established alternative to fat-trees are Clos-based topologies [42].



(a) Static DCN. **(b)** OCS-augmented DCN. **(c)** Full optical circuit switch (OCS)-based DCN.

Figure 3.2 Examples of static and reconfigurable DCN architectures. The black, solid lines indicate static links and blue, dotted lines are links to the optical switching fabric. The static topology (a) can either be augmented (b) or fully replaced (c) by an optical switching fabric. In the latter case, a two layer leaf-spine layout is a common choice.

The increasing demands and changes in traffic patterns and variability (cf. Chapter 2) have led to the observation that these static topologies become harder to operate in a cost-effective way [45, 52]. In the past decade, researchers started to explore topologies that integrate circuit switching elements, for instance by replacing parts of the static infrastructure. This creates hybrid topologies, in which, periodically or on demand, direct connections (or shortcuts) between parts of the network are established. In turn, these shortcuts reduce the requirements for switching capacity on the static topology part. Most research has converged to consider the ToR switch as connecting point to the optical fabric. However, there exist also notable exceptions from large operators [45, 66, 64]. Besides, prior work explored different concepts to augment [56, 50] or even fully replace the static topology parts (e.g., [52, 47, 53]). In all cases, the objectives are to maximize throughput between hosts and minimize the latency.

Figure 3.2 illustrates two basic architectures for DCNs. The first example (Figure 3.2b) is a hybrid topology, in which a traditional fat-tree topology (black) is augmented with an OCS (blue). The second example (Figure 3.2c) fully relies on optical switching for connecting the ToRs, no static fabric exists. In both cases, end-hosts connect via the packet-switched ToR to the network. The ToR has at least one optical transceiver which connects to the optical switching fabric. Several technologies exist to implement optical switching which we will discuss in the following.

3.2 Optical Switching Components

The main drivers of reconfigurable topologies are advances in optical switching components. Compared to electrical switching components, optical switches usually work on a circuit-level granularity agnostic to the used data rate compared to packet-level switching. The circuit-level granularity comes at the higher cost of setting up a circuit. However, switching speeds dramatically increased in the past years from tens of milliseconds to micro- or even nanoseconds [120, 53, 59]. This makes optical switching-based network architecture more viable compared to pure electrical packet switched ones. The following presents two major classes of optical switching devices: optical circuit switches and reconfigurable optical add-drop multiplexers.

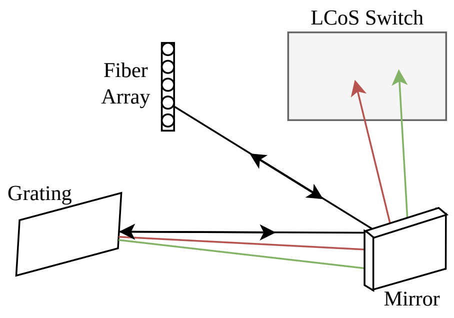


Figure 3.3 Liquid crystal (LC) on silicon wavelength selective switch (WSS). The figure is based on [119, Figure 6].

3.2.1 (Wavelength-selective) Optical Circuit Switches

OCSs have been established as the main optical switching component in the DCN context. They are available in a wavelength selective or wavelength oblivious fashion. In the wavelength selective version, colors can be forwarded differently. In general, a grating splits incoming light, and after passing lenses, the light is exposed to the actual switching element. Then, the switching element guides or reflects the light to the respective output port. There are several approaches to implementing the switching [119]:

- Arrayed waveguide gratings (AWGs) are used to separate several colors from a single fiber into multiple fibers. Combined with tunable transceivers, they have been used as optical switches [53]. By adjusting the color of the light, the egress port changes, and thereby, the connectivity is changed. The switching speed depends on the transceiver, but switching delays below 820ps were achieved [121].

- Microelectromechanical systems (MEMSs) are built using small mirrors that steer the light to the correct output port. Since the mirror must physically be moved, they generally show slower switching speeds compared to the alternatives. However, the reflection results in lower losses. MEMS-based OCS are commercially available at scales up to 320 ports, and these commercial versions provide switching speeds in the order of milliseconds [122, 123, 124].

- LC (on silicon) (Figure 3.3) modifies the polarization of the light in order to block or pass through to an egress port. First, a conventional grating splits the light onto a silicon chip. The chip is divided into small areas (cells) for each color. Electrical current is used to adjust the phase shift and steer the beam of the light to the intended output ports. LC can reach switching speeds in the order of microseconds [125].

- Rotating disks (rotor switches) are another option for implementing optical switching [48]. In contrast to the other two approaches, the configuration and connection cycle are fixed. Light is projected onto a rotating disk with holes that allow the light to pass to the egress port of the switch. Only a prototype implementation is available, reaching reconfiguration speeds of 20μs [120]

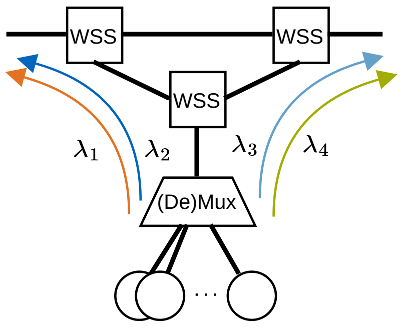


Figure 3.4 Structure of a 2-degree, colorless and directionless reconfigurable optical add-drop multiplexer (ROADM). Three WSSs steer the different colors ($\lambda_1, \lambda_2 \ldots$) from/to the add/drop ports or pass them through to the next node in the network. The (de-)multiplexer combines or splits different colors from/to the local ports (circles). Figure is based on [126, Figure 6].

3.2.2 ROADM

ROADMs are commonly used to set up circuits (or lightpaths) in the WAN context. They have passed several stages of evolution, adding more flexibility [126]. The basic functionality is to add or drop a specific wavelength to the local output port but let other wavelengths pass through to the next element. Initially, each ROADM had a pre-configured fixed wavelength to add or drop. In *colorless* ROADMs, the wavelength is flexible, e.g., configured by the connected transceiver. Further improvements include *directionless* ROADMs, which can send light from the local port flexibly on both uplink ports, and *contentionless* ROADMs, where different add-sources can use the same wavelength when they are forwarded in different directions in the network. This simplifies the wavelength assignment.

A ROADM is composed of a set of power splitters and WSSs that block or let different wavelengths pass. Figure 3.4 shows the structure of a 2-degree colorless and directionless ROADM. It consists of three WSSs and one colorless (de-)multiplexer. The multiplexer combines wavelengths from different input ports. The lower WSS guides the colors to the desired output port of the ROADM. The other two WSSs can pass through colors or steer them toward the drop ports.

3.3 Characteristics of Networks with Topological Reconfigurations

In order to better elaborate on the presented designs and solutions in this thesis, this section first briefly presents the components of a reconfigurable link (Section 3.3.1). These are also used to describe how different networking layers are affected or can be considered when reconfiguring the topology (Section 3.3.3). Lastly, Section 3.3.4 introduces the different classes of reconfigurations observed in the literature. We will later use these two properties to discriminate existing designs and solutions for the two use cases (Section 3.4 and Section 3.5).

3.3.1 Components of an abstract reconfigurable link

Figure 3.5 zooms in on the components of a single reconfigurable link. We ignore use case specifics and consider an abstract link. The link endpoints connect to the optical switching element via their transceivers and fibers. Such endpoints can either be hosts or (programmable) switches. Link endpoints generally both send and receive traffic. Therefore, the transceivers are

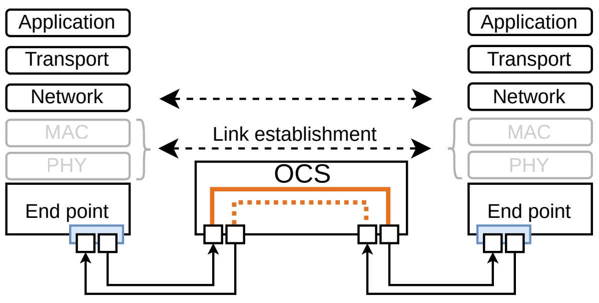


Figure 3.5 Components of an (abstract) reconfigurable link. Transceivers connect via the optical switching element. Arrows indicate possible synchronization efforts on the ISO/OSI layers.

bidirectional, and two fibers connect to the optical switching element (one for sending and one for receiving direction). At the OCS, one fiber connects to an ingress and the other to an egress port. With this differentiation, the configuration of the switching element, and thereby the topology configuration, can be represented by a bipartite graph of the ingress and egress ports. Each node can connect to exactly one other node forming a *matching*.[1] A topology configuration may contain bidirectional links, i.e., both directions of the transceivers of two endpoints are connected. In this case, the connections corresponding to the solid and the dashed line in the OCS in Figure 3.5 are set. Nevertheless, a large body of prior work and also this thesis generally assumes that circuits are unidirectional, e.g., only the solid line is set as circuit (cf. [46, 52, 47, 128, 48, 53]). Using unidirectional circuits intuitively leads to a higher degree of freedom when configuring the topology.

3.3.2 Reconfiguration Delay

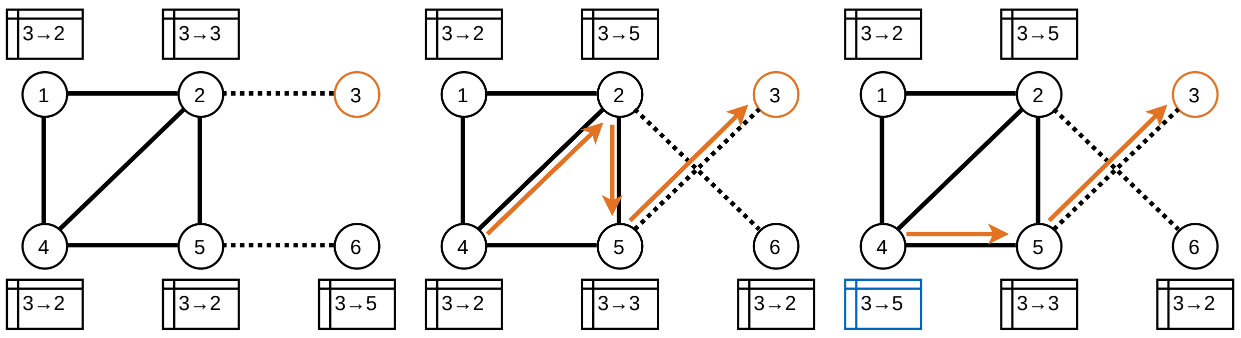
Each reconfiguration of such a link leads to an unavoidable interruption of the connectivity. The length of this interruption, the *reconfiguration delay* R, and the frequency of reconfigurations are fundamental factors for the efficiency of the reconfigurable topology. We define the time between two initiations of a topology reconfiguration as a *slot*. The length of a slot (s) is the time that the configuration is active plus the reconfiguration delay (R).

The reconfiguration delay is not only affected by the physical switching speed of the OCS but also by the behavior of the upper layers. Upon reconfiguration, i.e., establishing a new optical circuit, the lower layers of the ISO/OSI stack (PHY and MAC) must negotiate and establish the link. Afterward, the networking layer may need to exchange information about the logical connection and, moreover, propagate this information through the network (not shown).

3.3.3 Considered Networking Layers

A second characteristic of reconfigurable topologies is the set of actively considered layers for reconfiguration. Specifically, in Figure 3.5, the networking layer might exchange and forward information about the established link through the rest of the network. Besides the endpoints of the reconfigured link, redistribution of load to other links or nodes in the network turned out to be beneficial to utilize the full potential of the topological reconfiguration. Prior research

[1] In graph theory, a matching of a graph with vertices V is a set M of independent edges where every vertex in V is incident with one edge in M [127, Chapter 2]. From now on, the terms "matching" and "optical switch" are used synonymously throughout this thesis.



(a) Initial setting. **(b)** Only topology. **(c)** Topology and routing.

Figure 3.6 Example showcasing the benefits of considering multiple layers for re-optimization with reconfigurations. The solid lines show static links, the dotted ones dynamic links. The orange arrows show the forwarding path from node 4 to the destination node 3 using the entries in the forwarding tables (boxes). In Figure 3.6c, the blue entry has been updated. Updating the routing after the topological reconfiguration reduces forwarding path lengths.

contributes examples for both approaches considering different amounts and sets of layers (cf. Section 3.4.4).

To illustrate this, we consider Figure 3.6. The leftmost sub-figure shows the initial topology and IP routing configuration. Circles are IP routers and lines indicate logical connections. Next to each router is its forwarding table. The tables show only forwarding entries with respect to destination node 3. When only reconfiguring the topology but not the routing (Figure 3.6b), traffic from node 4 to node 3 will use a longer path than necessary (4⟶2⟶5⟶3), wasting the capacity of the network. Note that forwarding tables of the nodes that are endpoints of a reconfigured link (here: node 2 and 5) must always update their forwarding table. When considering multiple layers for reconfiguration (Figure 3.6c), the adapted solution updates the forwarding entry at node 4 (blue highlight) and sends traffic from node 4 to node 3 via the shortest possible path (4⟶5⟶3). Thereby, less capacity in the network is used, i.e., the available resources are used more efficiently. This example only considers topology and routing. However, an extension to demand- or application-related layers is possible.

3.3.4 Classes of Topological Reconfigurations

A third attribute of reconfigurable topologies is the reconfiguration class. In this thesis, we define the reconfiguration class by the fact that the reconfigurations are demand-aware or demand-oblivious, which both are observed in prior works [117]. For *DA* reconfigurations, the network controller gathers some demand estimate of the network, e.g., a (predicted) demand matrix. Based on this matrix, the controller optimizes the required optical links. Figure 3.7a illustrates an example of *DA* reconfigurations. Each column shows the OCS configuration and the demand matrix of the respective time instance. The first two demand matrices are the same, and so are the two optimized configurations of the OCS directly connecting the source and destination nodes with high demand (orange/shaded areas). The demand pattern changes for the following time instances, and the OCS's configuration is adapted accordingly. In general, this class of reconfigurations results in large reconfiguration periods as demand collection or estimation is a complex problem on its own [48]. Moreover, it requires optical switches that can set links on request. That is *rotor* switches are generally not usable.

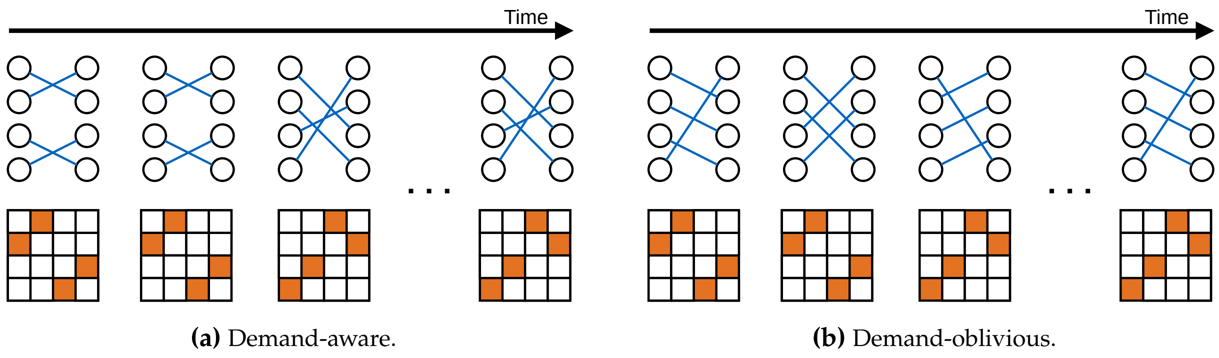


Figure 3.7 The two classes of topological reconfigurations: demand-aware (*DA*) and demand-oblivious (*DO*). The upper row shows the topology configurations and the lower row the demand matrix. *DA* changes the topoloy according to the demand, whereas *DO* follows a pre-defined sequence regardless the demand matrix.

The second class, *DO* reconfigurations, does not rely on demand estimates. Instead, it follows a pre-defined, fixed schedule to configure the circuits. The example in Figure 3.7b shows how the OCS's reconfiguration changes independently of the demand. The advantage is generally shorter reconfiguration periods but at the cost of reduced flexibility. *DO* reconfigurations can be implemented with all types of optical switches.

3.4 Reconfigurable Wide Area Networks

The adaptation of WANs to changing, mainly growing, traffic demands has been well explored by the research community [129, 130, 119]. Much of the existing work considers long-term capacity planning for the operation of the network throughout its lifetime. Here, the reconfiguration periods are in the order of months or years. Another share of prior work considers more frequent reconfigurations with reconfiguration delays in the order of minutes [57]. Most recent work achieved reconfiguration delays in the order of seconds in a testbed environment [131]. These works build on applying software defined networking (SDN) to optical transport networks. A more in-depth introduction is found in [132] and [119].

This section briefly summarizes essential aspects that have been explored by previous work, with a focus on how reconfigurations and different layers have been accounted for. Therefore, it lists and gives examples of common modeling and optimization approaches to minimize the operating costs when reconfiguring WANs by means of adding or removing lightpaths and optimizing routing (Section 3.4.1). It continues with works that explicitly try to minimize reconfigurations (Section 3.4.2). With recent technological advances, bandwidth-variable transceivers (BVTs) became feasible and have been explored to increase the efficiency of resource utilization (Section 3.4.3). Finally, we overview how explicit application requirements or requests have been integrated into the reconfiguration of WANs (Section 3.4.4). This illustrates how prior research has made attempts to extend the layers considered during reconfiguration [119].

3.4.1 Minimizing operating costs

The main objective of optimizing the logical IP topology and routing revolves around reducing the OPEX. The definition of OPEX varies, but commonly considered components or proxy metrics include the amount of deployed capacity in the IP layer, the number of used transceivers, or the power consumption. The general setting is to minimize the costs while serving a given demand. Therefore, the metrics above are also referred to or measured as efficiencies.

Early work on reconfigurable WANs refers to the adaptation of the logical topology over time. Namely, examples are [133, 61]. These works focus on the IP topology and consider the routing adaptation only as a side product. For instance, Gencata and Mukherjee [133] propose a Mixed Integer Program (MIP) to minimize the most loaded link in the logical topology. They propose a periodic measurement of the link utilizations with a threshold-based decision to trigger the re-optimization. For the optimization, they use a similar model as Ramamurthy and Ramakrishnan [61], who focus on the average packet latency and the resources used (total lightpaths set up and physical fibers used). Besides the logical topology, both works implicitly optimize the IP routing.

Other works take the notion of several layers more explicitly into consideration [134, 135, 136]. Akgun and Buzluca [134] present an optimization that explicitly considers the ratio of applied traffic grooming. That is the ratio of dedicated lightpaths between node pairs and bundling traffic on the IP layer by means of IP routing. Lopez et al. [135] propose an approach in a similar direction. They design an algorithm using Bayesian decision theory to decide if optical bypasses (dedicated lightpaths between node pairs) or grooming should be used to fulfill a traffic demand. Specifically, the used risk function trades off the gains from the bypasses with the used resources in the physical topology. Morales et al. [136] use traffic predictions to facilitate reconfiguring the network a priori, i.e., before the demand changes. This contrasts the previously presented works, which decide based on traffic measurements. The actual optimization problem minimizes the used transceivers and the amount of unserved traffic.

Another stream of research models and integrates energy consumption of networking equipment as operating cost into the optimization of the reconfigurable WAN [137, 138, 139, 140]. An initial comparative study is given by Idzikowski et al. [137], who benchmark the energy savings when reconfiguring only the IP layer, only the optical layer, or both. Their evaluation demonstrates that the largest energy savings stem from reconfigurations in the IP layer. Van Heddeghem et al. [138] follow up on this by providing a more detailed model of the power consumption. In addition, they demonstrate the major benefits of using optical bypasses on energy savings. Besides energy savings, these two works consider only fulfilling the demand. In contrast, Lee and Rhee [140] focus on a different aspect. They relate the energy savings to the increase in delay of the transmissions by means of end-to-end propagation time. Their evaluation describes the trade-offs between these two metrics. More recently, Fenz et al. [141] evaluate the benefits of reconfigurable WANs under varying traffic patterns. To this end, the authors compare a joint logical topology and traffic engineering (TE) formulation against a topology oblivious baseline, using publicly available demand and physical topologies. Finally, they present an improved *DO* topology design scheme to account for reconfiguration costs.

3.4.2 Minimizing reconfigurations

The previously listed works all consider the adaptation of the IP topology. But only a few of the examples consider the number of reconfigurations during their evaluations or contain triggering mechanisms to limit the frequency of reconfigurations, e.g., [133]. This part gives an overview of prior work that explicitly addresses the impact of reconfigurations during optimization or focuses on the stability of the IP and optical routing.

Much of these related work revolves around the impact of optical network reconfiguration on routing stability. For instance, Sinha and Murthy [142] present a framework to evaluate the trade-off that reconfiguration policies provide with respect to efficient resource utilization and minimizing traffic disruption. Moreover, they propose a policy that leverages predictions of the future demand matrices to pre-plan the reconfigurations. Bianco et al. [143] explore the Pareto frontier of minimizing deployment and reconfiguration costs. They first study the characteristics of the Pareto-optimal solutions and then present different reconfiguration schemes to allow operators to explore the trade-off of the two costs. Tran and Killat [144] consider the problem from a different direction. They extend Gencata's work [133] and propose a reconfiguration policy based on a load-balance indicator. They further present a Mixed Integer Linear Program (MILP) that jointly minimizes the total number of lightpath changes as well as the average path length over all node pairs. Bonetto et al. [145] integrate reconfiguration costs into the optimization of the IP-over-OTN network. They present a MIP formulation and three heuristics that solve the problem on a per time instance basis. The goal is to minimize energy consumption and reconfigured traffic. The latter is also considered by Ohsita et al. [146], who evaluate how gradual reconfiguration using estimations of the future traffic performs.

While the aforementioned works consider mainly the addition and removal of lightpaths as reconfigurations, other works look more specifically at the impacts on the routing. An example is given by the works of Chamania et al. [147, 148], who specifically focus on keeping the reconfigurations in the IP routing layer low. To this end, they propose an algorithm to add optical bypasses only over congested links. These bypasses are not considered when updating the general routing but only to offload traffic from the congested links. They present an Integer Linear Program (ILP).

3.4.3 Reconfiguring modulation and bandwidth

Early works in the domain of reconfigurable WANs are limited to adding or removing lightpaths in order to add new logical connections or increase the capacity of logical links. With advances in optical devices, elastic optical networks (EONs) became available. EONs use BVTs which allow adjusting the link's bandwidth using a single lightpath, e.g., by increasing the used range of frequencies (bandwidth) or changing the modulation scheme. Note that this can generally impact the performance of other lightpaths traversing the same fiber [149]. With the new opportunities, the range of parameters increases, but the considered optimization objectives still revolve around minimizing operating costs and reconfigurations. Tanaka et al. [150] give a first particular example. They provide a multi-period IP-over-EON reconfiguration algorithm minimizing the overall power consumption. However, they consider large periods between reconfigurations, e.g., years or months.

More recent work includes [151, 152, 153, 154]. RADWAN [151] builds on BVTs and adapts the optical links' capacity based on the observed signal-to-noise ratio (SNR) to achieve higher throughput while also considering the routing churn during reconfigurations. To this end, the authors formulate an extended multi-commodity flow problem that maximizes throughput while also considering the churn in the network. Operators can tune the trade-off between throughput and reconfiguration. Besides a simulation-based evaluation, RADWAN is also shown to be feasible in a testbed implementation. Another instance is provided by Zhong et al. [155]. Like

RADWAN, the authors argue that the length of the optical bypasses may reduce the achieved capacity. In turn, they present an algorithm to split optical lightpaths into shorter ones that allow using higher modulation schemes. Thereby, the capacity of individual links is increased. However, at the cost of traffic interruptions. Tseng [156] focuses on the planning reconfigurations in the scenario created by RADWAN. He explicitly considers the reconfiguration delay and presents a multi-step planning algorithm with limited reconfigurations per step. Furthermore, it is shown that the optimal solution to the problem is NP-hard. Therefore, a Linear Program (LP) heuristic is derived. Shoofly [152] extends RADWAN to the optimization of optical bypass connections. The authors leverage data from a cloud provider to argue that optical bypasses come with longer optical paths before the re-generation of the signal. This potentially reduces the usable modulation schemes and resilience to link failures. Shoofly bases on a MIP to allocate the demands to the optical bypasses considering the capacity constraints stemming from the path length. RADWAN's approach is also extended along other dimensions. Jia et al. [153] consider it as a basis for the online transfer scheduling problem and present several greedy and online scheduling algorithms that are shown to provide several competitive ratios for optimization objectives, such as makespan and minimum sum completion time for arbitrarily sized jobs. Their work is further extended to the sum of flow times in [154].

3.4.4 Considering special demand types or application requirements

Finally, also the optimization of reconfigurable WANs with different types of demand has gathered some attraction; for instance, in the case of network function virtualizations (NFVs) or virtual network embedding (VNE). Here, the demands are not given as matrices of (IP) demand but rather as virtual network (VN) requests with specific bandwidth requirements that must be met by allocating virtual nodes and edges on the physical (substrate) topology. The introductions and surveys in [157, 158] provide an in-depth overview of existing solutions in this context.

Some notable examples are [159, 160, 161, 162, 163, 164]. Shakya et al. [159] present an embedding algorithm for VNs in EONs. They particularly address the problem of fragmentation of the resources in the EON. The presented solution prioritizes the allocation of consecutive blocks of resources, i.e., wavelengths. The authors also present a reconfiguration algorithm to reduce the alignment of resources after VNs have been removed from the substrate network. Nonde et al. [160] combine VNE over EON with a focus on energy consumption. Their modeling covers several aspects of power consumption from the IP and optical layers and is integrated into a MIP formulation with two different objectives. These objectives cover the total substrate resources allocated to a request as well as the number of used (active) substrate links and nodes as a proxy metric for energy consumption. To give a better understanding of the impact of the allocation of VNs on different layers, i.e., IP or optical, Zhang et al. [161] propose an auxiliary graph model. They evaluate the performance in three classes of substrate topologies: electrical layer, optical layer, and multi-layer. Chowdhury et al. [162] follow a similar direction, highlighting the benefits of multi-layer networks for VNE.

Application-awareness is also introduced to the multi-layer optimization regime by other means [63]. For instance, Ghonaim et al. [165] present a design where applications can request optical bypasses (shortcuts) from the network. The idea is to use this on-demand connection to alleviate the bursty traffic behavior of applications. They evaluate their approach using a simulation and demonstrate an efficiency increase of 33%. The routing is not optimized since the bypass is reserved for the single node pair or application. Other works consider application requirements such as latency constraints during optimization [166, 167, 168, 62, 63, 169]. On the one hand, these works increase the span of layers that are integrated during optimization but, on the other hand, do not specifically cover intermediate layers. For example, Klinkowski and Walkowiak [166] optimize for a mix of anycast traffic from content delivery networks (CDNs) and fixed source destination generic traffic in an EON setting. They compare settings with different settings of flexibility with and without BVTs and varying modulation capabilities to illustrate the gains of EON for such demand allocation. Perello et al. [167] consider optimizing anycast demands in a CDN setting. However, they do not include the IP layer in their optimization. They present a MIP formulation and also a heuristic to reduce the time to solution.

Lopez et al. [168] and Rozic et al. [62] take a more implementation-oriented view. They introduce frameworks to implement application-aware IP-over-WDM networks. That is, they define the necessary interfaces between the application and the control plane of the network. Finally, Luo et al. [169] present an optimization scheme, DaRTree, for multicast transfers that considers the jobs' deadlines. DaRTree uses a linear program relaxation and deterministic rounding to adapt the topology and routing configuration specifically for multicast jobs. The evaluation shows that it can increase the number of accepted requests by 70%.

Takeaway: Reconfigurable WAN have been subject to a wide range of research touching different problems and covering a multitude of optimization aspects. The previous overview highlights that existing work largely covers multiple layers for optimization and reconfiguration. However, we observe that set of considered layers is limited, and that joint consideration of the layers spanning from optical via IP to the demand layer and the related resource optimization problem is missing.

3.5 Reconfigurable Datacenter Networks

The literature contains a solid amount of prior work also for the use case of RDCNs. The proposals differ from the WAN use case as they consider shorter reconfiguration periods, mostly within the sub-second range. The existing RDCN proposals vary in the used technology, achieved reconfiguration speeds, and on an algorithmic level. This section starts with an overview of *DO* RDCN proposals in Section 3.5.1 followed by *DA* (Section 3.5.2). Moreover, it describes what has been explored with respect to higher layer protocols, e.g., the transport layer, which are not designed for frequently changing network conditions (Section 3.5.3). Finally, Section 3.5.4 reports on theoretical results achieved in prior work. For more detailed surveys, the reader is referred to [119, 118, 170].

3.5.1 Demand-oblivious RDCNs

A pioneer work for demand-oblivious RDCNs is RotorNet [48]. The authors explicitly decouple the switch configuration from the traffic patterns in order to omit statistics collection and a centralized control plane. The basic idea is to cycle through a fixed set of matchings to emulate a full graph and have a link between every pair of racks once in every cycle. While this approach performs well for uniformly distributed demands, skewed demands deteriorate efficiency. Therefore, RotorNet forwards skewed demands indirectly to an intermediate rack similar to Valiant load balancing (VLB) [171]. The presented control plane protocol, RotorLB (RLB), ensures that racks are not overloaded by indirect traffic and, thereby, guarantees transmission to the final destination within two hops and two full cycles. Opera [49] advances the core idea of RotorNet. The matchings are constructed so that they resemble a time-varying expander graph. Moreover, the OCSs change their matchings one by one so that the network is connected at every point in time. Opera divides traffic into low-latency and bulk. The former uses multi-hop routing on the temporarily static expander graph with NDP [172] as the transport protocol. The bulk traffic is scheduled with RLB. The evaluation demonstrates reduced latency compared to RotorNet and static expander-based and Clos-based topologies of similar sizes.

Sirius [53] proposes an all-optical, passive core network that consists only of a single layer of AWG routers. The switching capability moves to the transceivers, which are designed to adjust their sending wavelength quickly. The custom design of wavelength tunable lasers and clock synchronization mechanism allow cycling at reconfiguration periods of hundreds of nanoseconds. Like RotorNet, Sirius periodically cycles through a set of matchings and uses traffic indirection to create a uniform demand pattern. However, it uses a randomized approach, which converts the demand matrix into an uniform matrix. The smaller slot size can fit only a single packet. The authors demonstrate the feasibility of Sirius and show its cost and performance improvements compared to (non-blocking) static topologies.

MARS [173] addresses the high buffer requirements of previous RDCNs such as RotorNet, Opera, or Sirius, which cycle through the complete graph over time. The authors conclude that such topologies are infeasible at scale and identify alternative solutions that periodically cycle through graphs with smaller nodal degrees. MARS chooses the nodal degrees to optimize throughput subject to given available buffer sizes and delay tolerances.

3.5.2 Demand-aware RDCNs

The majority of RDCNs considers *DA* reconfigurations. First, the overview summarizes ToR-level RDCNs that use heuristics or optimal solutions to find configurations for single demand matrices. Later, it lists works that use scheduling and decomposition approaches as well as machine learning (ML)-based optimizations. Further, we can distinguish topologies with segregated and non-segregated routing and summarize pod-level and application-specific designs.

3.5.2.1 Matching-based reconfigurations

A large group of RDCN designs tries to find the best topology configuration for a given demand matrix. We refer to these classes as "machting-oriented" solutions.

A first example is Helios [45]. It is one of the first hybrid topology designs that combine an electrical packet-switched network with an optical circuit-switched one to augment the electrical network with direct connections in the optical domain. The control loop collects traffic statistics (as a flow-rate matrix) and classifies pod-to-pod demands into mice and elephant flows. After removing mice flows from the matrix, it uses Edmond's algorithm to determine the new configuration of the OCS. Edmond's algorithm serves for optimization in c-Through [56] too. C-Through proposes a host-based approach for routing over a hybrid packet and circuit-switched network. The hosts add a special Virtual LAN (VLAN) tag to every packet thereby controlling its routing. The ToRs forward the packet according to the VLAN tag over the packet-switched or the circuit-switched part. To leverage the higher bandwidth of the optical network, large Transport control protocol (TCP) buffers are configured (so that TCP can increase the congestion window more quickly). The discussion also covers the problems of circuit reconfiguration delay (the minimum time needed to set a circuit for every rack of interest) and synchronization between applications concluding that c-Through is more suited for applications with loose or no synchronization.

Proteus [174] employs a different approach. It uses a fully optical interconnect between the ToR switches. It combines MEMS-based optical switching with WSS to adjust the connectivity and capacity between ToRs. The optimization bases on a greedy heuristic. The authors only give a primer on costs and power consumption but leave out a detailed performance analysis. OSA [46] builds on the core idea of Proteus to combine optical space switching with adjusting link capacities. It features the same components and optimizes the configuration in three steps: OCS configuration, routing configuration, and wavelength assignment. It is shown to outperform Helios and c-Through on several traffic patterns. Megaswitch [54] is another example of a design using WSSs. It builds around a ring of multiple parallel fibers. Circuits are set up using different wavelengths along the ring. The controller ensures basic connectivity and dedicates more wavelengths, i.e., more capacity, to node pairs with high demands.

Flat-Tree [51] proposes the idea to dynamically (and also only partially) convert the DCN between a Clos-like topology with more efficient wiring and good intra-rack performance and a random graph with superior throughput. The approach employs a decentralized scheme that allows different zones of the network to be connected differently. Routing is done with k-shortest paths and multi-path (MP)-TCP. The authors address how to implement the control plane using SDN and how to minimize the number of forwarding rules. No explicit method of triggering the conversion is mentioned, but the authors refer to network upgrade approaches. The evaluation in simulations and a testbed demonstrate that Flat-Tree performs competitively to an LP optimal solution and can adapt to workloads with varying locality (global, pod, rack). However, the exact method of triggering the adaptation is not mentioned. Flexspander [175] builds on the idea of a flexible *Expander* network. The intra-pod topology is a fixed *Expander*, and OCSs provide dynamic connectivity on the pod level. This increases the capacity towards highly skewed traffic demands. The authors present a MIP and heuristics to optimize the OCS layer and evaluate their topology with packet-level simulations on several traffic patterns.

Other works propose to use wireless technology or free-space optics as direct interconnects between ToRs. Initial work with wireless links is presented by Zhou et al. [176]. They propose to use beamforming to reduce interference and to leverage reflections at the ceiling to create additional paths for the beams. A second example is FireFly [177]. Besides presenting the hardware implementation for the free-space optics, the authors introduce a new metric to measure the performance of the dynamic topologies. The so-called dynamic bisection bandwidth is the minimum bisection bandwidth given the optimal topology for each possible network partition. They further introduce optimal (ILP-based) and heuristic solutions to the reconfiguration and the routing problem. Finally, they address the issue of correctness during the reconfiguration: avoiding black holes, maintaining connectivity, and upper-bound the packet latency. The proposed system leverages the SDN concept, is implemented for packet-level and flow-level simulation and emulation, and is evaluated for several kinds of traffic patterns (uniform, hotspot). ProjecToR [47] is another RDCN using free-space optics. It is a fully reconfigurable topology that is split into two parts. The first part is reconfigured on a large time scale, e.g., once per day, to set up a basic fixed topology that provides connectivity but is also optimized toward the demand. The remaining available transceivers are used for opportunistic scheduling of *DA* links. When senders accumulate a certain amount of packets towards a destination, they request to set up a link. The paper proposes a distributed algorithm solving the stable marriage problem. The approach has been extended from an algorithmic point of view in [178], focusing on minimizing flow completion times. The authors present a stable matching algorithm that is shown to be $O(\epsilon^{-2})$-competitive.

A series of works considers RDCNs from the perspective of adjustable data structures [179, 180, 181]. For instance, Splaynet [179] uses distributed binary search trees as the basis for minimizing routing costs in RDCNs. The authors formally evaluate the performance and guarantees of their algorithm and prove that it is optimal if specific criteria are met. Finally, they extend their approach to multiple trees. Similar works [181, 180] also build upon self-adjusting (tree) data structures as optimization approaches for RDCNs.

C-Share [182] tackles the problem of elephant flow routing in RDCN networks by routing large flows from different source-destination pairs over common (existing) circuits, i.e., re-using circuits, and allows non-segregated routing. That is, a flow can use both electrical (static) and optical (dynamic) links when forwarding between ToRs. The design uses a flat ToR-level topology and host-based elephant flow detection and tagging to reduce the number of forwarding rules on the ToRs. The proposed heuristic runs periodically and greedily tries to offload as much traffic as possible to the circuits. It uses a product of demand and path length to calculate a weight which is used to calculate the matchings. For shared circuits, the offload happens only on the last hop. The approach is evaluated using both simulation and emulation and shows savings in deployed routing rules and flow completion time for mice as well as elephant flows. A completely different structure is employed by Wang et al. [183]. They propose to add OCSs between servers and ToRs. The rationale is to balance traffic from the servers across the ToRs so that aggregation and core layers are more evenly loaded and bottlenecks are relieved. Packet-level simulations show the advantages of such a design compared to an oversubscribed network and higher power efficiency compared to a non-blocking network.

Finally, there is a stream of prior work that considers a different (explicit) optimization goal [184, 185, 186]. The proposed topologies split the spine layer into switches dedicated to intra-pod and inter-pod traffic. While optimization is also performed with respect to one snapshot of demand, the performance is assessed regarding packet loss and latency. That is, the topology accommodates all demand (given as rates) by adapting circuits and allocated capacity (modulation). However, these works do not assess how more frequently used metrics, such as flow completion time or throughput are affected.

3.5.2.2 Demand-aware scheduling RDCNs

Aside from optimizing a single topology configuration for a given demand matrix, other works have considered scheduling a sequence of topology configurations to serve the demand matrix. Thereby, these approaches present one way to consider reconfiguration times.

Mordia [52] names two major factors that limit reconfiguration speed and hence deteriorate the benefits in existing RDCNs: slow OCS and slow decisions in the control plane due to softwareization and hence, they propose a low-delay circuit switch ($\approx 11.5\mu s$) and a scheduling scheme that leverages short-term traffic predictions and application-level information. Calculating the schedule, i.e., a sequence of circuits, allows decoupling calculation from actual switching. After collecting and scaling the demand matrices such that all row and column sums are $= 1$, Mordia uses the Birkhoff-von Neumann decomposition (BvN decomposition) to find the schedule of configurations. The schedule is pruned to schedule only circuits with a minimum duration. The goal is to keep the duty cycle as high as possible. Thereby, the amount of traffic sent over the optical network can be varied. ReacToR [50] builds on Mordia's design but extends to a hybrid network with electrical (static) and optical topology parts. Moreover, it builds on the observation that segmentation offloading to the network interface card (NIC) leads to bursts with a duration of tens to hundreds of microseconds. The idea is to circuit switch according to these bursts to hide reconfigurations from the transport protocols. It presents data and control plane design to schedule the demands in such a hybrid network. A prototype implementation and extensive simulation study demonstrate feasibility and performance improvements.

Solstice [187] formalizes a hybrid switch model, which consists of parallel OCS and electrical packet switch with a lower rate but flexible forwarding. The authors present an ILP to optimally solve the scheduling problem and a heuristic algorithm. The heuristic builds on the fact that bi-stochastic matrices are suited for BvN decomposition. Therefore, it first adds artificial demand to convert the demand matrix into a bi-stochastic matrix and then employs a loop similar to BvN decomposition. The loop terminates when the leftover demand is small enough to be transmitted in parallel to the schedule via the packet switch. Eclipse [188] uses the same hybrid switch model but improves upon Solstice. The presented algorithm is a constant factor approximation of the optimal solution.

Vargaftik et al. [128] extend the hybrid switch model by interconnecting the switches and creating composite paths with electrical and optical components. The goal is to improve performance for one-to-many and many-to-one traffic patterns. In the previous model, those patterns are either constrained by the packet switch's low bandwidth or the OCS's high reconfiguration costs. The scheduling algorithm is inspired by existing approaches to scheduling in hybrid networks. It extends the demand matrix by adding entries for potential one-to-many and many-to-one demands to the rows/columns that correspond to the inter-fabric link. The demand matrix is fed to the hybrid switch scheduler, and the output is split up for the different fabrics to create the final schedule. The evaluation considers switches with different reconfiguration costs and shows that the new proposal outperforms Solstice and Eclipse.

Sunflow [189] presents a different extension of the hybrid switch model: it does not interrupt all circuits upon reconfiguration (not all-stop). The scheduling algorithm, which is specifically designed for coflow traffic patterns, uses an event-driven mechanism to reconfigure when at least one circuit becomes idle. This approach bases on the assumption of knowing the amount of data that every sub-flow of a coflow transmits and reduces the penalty due to reconfiguration.

3.5.2.3 ML-based optimization

Given the hardness of the scheduling problem, some work explored ML to optimize the topology or at least to speed up the optimization process [190]. The first example is xWeaver [191]. The paper proposes a Supervised Learning based solution to topology adaptation. xWeaver has two phases. During the offline phase, the system uses demand traces and flow-level simulations to create and learn mappings between demand, topology configuration, and performance, e.g., flow completion time. This learned mapping serves as input to a heuristic algorithm that searches for high-performance topologies given a demand matrix. The generated dataset of high-performance demand-topology pairs serves as input to train another Neural Network (NN) that maps from demand to topology configuration. This NN is used during the actual deployment phase to predict topology configurations. DeepConf [192] uses Reinforcement Learning to optimize the topology configuration. The whole network is controlled by an SDN controller that runs multiple Reinforcement Learning agents for different tasks (routing, topology reconfiguration, energy saving. . .). The DeepConf framework provides an abstraction of the network that the agents use for training on data from domain-specific simulators. Specifically, the agent predicts the topology configuration from an observed demand matrix. For each link, the output layer of the NN gives probabilities of the potential optical links to be used. The links with the highest probabilities are then selected. The generated reward represents the link utilization and flow completion time as performance metrics. The evaluation compares to the optimal solution on Fat-Tree [42] and VL2 [193] and shows that DeepConf performs comparably to the optimal solution while having shorter times to solution in the online case.

3.5.2.4 Pod-level RDCNs

The works above consider reconfiguration of the topology at high frequencies, i.e., in the order sub-second reconfiguration periods. A different stream of research revolves around a more robust topology optimization to increase the reconfiguration periods to the order of hours or even days [194, 66, 64, 195]. These topologies consider pod-level RDCN so that the re-optimizations are performed on more aggregated traffic which has been shown to generally be more stable [66]. Specifically, METTEOR [194] is a first example of such robust optimization. The idea is to optimize the topology based on representative samples of the observed demand. Therefore, a sequence of observed demand matrices is first clustered. The found cluster centers serve as input for a robust optimization that maximizes the minimum throughput across the cluster centers. In follow-up work, they extend the approach to perform robust optimization over a convex set of critical traffic matrices [195]. Zhang et al. [66] also consider a pod-level RDCN to benefit from higher traffic aggregation. They argue that ToR-level designs are not feasible due to the unavailability of suitable equipment (e.g., OCSs with high port count) and the necessity of detailed traffic predictions. The system reconfigures based on a convex hull of demand predictions while minimizing the maximum (logical) link utilization. Finally, Poutievski et al. [64] report on deploying an RDCN in a production environment. They leverage pod-level reconfigurability, and use a reconfiguration period of several hours. Generally, the routing is limited to 2-hop paths, but specific details on the optimization are left out. The trade-off between ToR-level and pod-level RDCNs is explored in more detail by Teh et al. [196]. The authors compare both types under various conditions and with *DO* and *DA* reconfigurations with respect to power, cost, and performance.

3.5.2.5 Application-specific RDCNs

Application-specific designs or reconfiguration algorithms have been introduced along two streams. The first stream considers coflows, i.e., groups of flows that relate to each other [128, 189, 197]. Namely, examples already discussed are cp-Switch [128] and Sunflow [189]. Another example is SplitCast [198] which is specifically designed for multicast traffic. Here, the differences are mainly in the considered objective function and preemption behavior.

A second specific application that is more in focus by the research community is distributed machine learning (DML) [103, 199, 200, 201]. A first example is PSNet [103] which proposes an RDCN specifically designed for parameter server (PS)-based DML traffic. The authors argue that the network connection of parameter servers becomes the bottleneck during training since all worker nodes have to synchronize with this single point. Accordingly, the PS should be provided with more bandwidth. PSnet has two levels: The lower level (modules) contains several ToRs, PSs, and worker nodes. The PSs are connected to all ToRs in a module (using multiple interfaces), and workers connect through a circuit switch to ToRs for resilience. On the upper layer, modules form an R-regular graph where R is the number of ToRs per module. Ideally, PSs and workers of a job are allocated in a single module to keep paths short. The topology configuration is optimized when a new training job starts. SiP-ML [200] proposes an optical interconnect using silicon photonics as switching elements. It interconnects graphics processing units (GPUs) in a ring and uses WSS to set up circuits. Besides optimizing the topology, it partitions the training job across the workers considering data and model parallelism. The OCSs' configuration is optimized using an ILP and a heuristic. TopoOpt [199] extends the basic idea from SiP-ML and presents a joint topology and workload allocation. The optimization follows an iterative process optimizing the workload given the topology and vice versa until the solution converges or a maximum number of iterations has passed. Since the traffic pattern of a single job is fixed throughout the training process [2], the OCSs are configured once per job and kept until the job finishes. The authors show that their TopoOpt can allocate multiple jobs simultaneously.

3.5.3 Higher Layer Considerations for RDCNs

Looking at the higher networking layers, such as IP routing and the transport layer, significantly fewer works have addressed these explicitly. An exception for a (generic) routing solution for RDCNs is TAGO [202]. TAGO is designed for block(pod)-level reconfigurable topologies. It prioritizes direct global paths and balances traffic among available direct interconnects with per-packet decisions. For intra-pod routing, equal cost multi-path (ECMP) is used. For inter-pod traffic, it randomly chooses a boundary router with probability proportional to its capacity to the destination pod. To implement this, TAGO uses an SDN controller, which installs the necessary rules with a global view. The authors demonstrate that TAGO increases throughput and reduces tail latency for several traffic patterns and topologies compared to ECMP, VLB and baselines that consider shortest path routing. However, from a practical point of view, the churn in the routing table and the achievable reconfiguration periods still need to be clarified.

Abu-Tair et al. [203] perform an emulation-based study of performance improvements due to RDCN. They focus on the throughput increase and completion time decrease when using TCP. The study compares relatively simple flow scheduling techniques and circuit assignment algorithms and only evaluates the throughput and the completion time but does not elaborate on additional traffic due to reconfigurations. They observe performance increases only when using *DA* scheduling schemes. ReTCP [204] studies the behavior of TCP reconfigurations with heterogeneous capacities for electrical and optical links. The authors illustrate that existing TCP variants cannot sufficiently grow the congestion window when the high-capacity optical link becomes available. To alleviate this, ReTCP considers explicit notifications before the circuit is set. Triggered by the notifications, the end-hosts can increase the congestion window. A presented alternative solution is to dynamically adjust the sizes of the ToRs' virtual queues to pre-buffer packets. Time-division TCP [205] extends on the idea of explicit notifications of circuits. It maintains an independent congestion state per distinct path in the reconfigurable fabric, e.g., one for the path via the electrical (low capacity) part and one for the path via the optical (high

[2] An observation we also made in Chapter 2 of this thesis.

capacity) topology part. Connection-wide tasks are organized in a shared way. Time-division TCP is evaluated on an emulated network and shows significant performance improvements over previous variants such as datacenter TCP or multi-path TCP. PowerTCP [206] considers a different approach. Although it is not directly tailored towards RDCNs, it achieves high circuit utilization and significantly reduces tail latency. To do so, it relies on in-band network telemetry to estimate the bandwidth-delay product and to quickly react to changes.

3.5.4 Performance Bounds for RDCNs

Several works explore the RDCNs from a more theoretic point of view, formalizing the problem, deriving performance bounds, and assessing the complexity of the related problems [117, 207, 190, 208]. Foerster et al. [207] study the algorithmic hardness of hybrid RDCN. They prove that commonly used matching algorithms are optimal when there is only one reconfigurable switch in the topology and the routing is enforced to be segregated, i.e., traffic is routed either via the optical or the electrical network. Furthermore, they show that the topology and routing optimization problem becomes NP-hard when non-segregated routing is allowed. In [190], Foerster et al. extend their work with a focus on non-segregated routing. They demonstrate that non-segregated routing policies for RDCNs can be computed in polynomial time under specific assumptions. However, variations of the problem that relax these assumptions are proven to remain NP-hard.

Zhao et al. [208] present a metric to capture the performance of a physical topology given a demand. The metric relies on the achievable throughput and relates the value of the given topology to those of any topology (including the "best"). Thereby, the metric can be used to capture the optimality of a physical topology given the demand. The metric is used to prove the performance guarantees of four representative RDCNs. Amir et al. [209] study the performance bounds and design of optimal RDCNs, focusing on the trade-off between maximizing throughput and minimizing latency. Specifically, given a constant throughput rate, they identify the minimum achievable latency of a design that satisfies the throughput constraint, i.e., they obtain a lower bound on the maximum latency. This analysis is limited to specific network sizes. In follow-up work, the authors extend their results to networks of all sizes [210].

Takeaway: Previous work covers many aspects of RDCNs, including proposals using the different classes of topological reconfigurations. However, we observe that practical designs of RDCNs, particularly with non-segregated routing, are missing. Moreover, joint consideration of both reconfiguration classes and their interplay within a single topology has yet to be studied.

3.6 Unified Model for RDCNs: ToR-Matching-ToR

By carefully inspecting existing proposals of RDCNs, we observe that most of them (*DA* and *DO*) employ a two layer leaf-spine architecture. We can generalize this architecture into a model with n leaf switches and k optical spine switches. In order to account for the *DA* and *DO* topologies, the spine switches can be of different types: static (ß), *DO* (Dobl) and *DA* (Da). From each switch type, there are k_s, k_{do} and k_{da} switches, respectively with $k = k_s + k_{do} + k_{da}$. Each optical spine switch connects each of its ingress-ports to exactly one egress-port and vice versa forming a *matching* (cf. Section 3.3.1). The resulting structure is a ToR connected to a matching to a ToR and

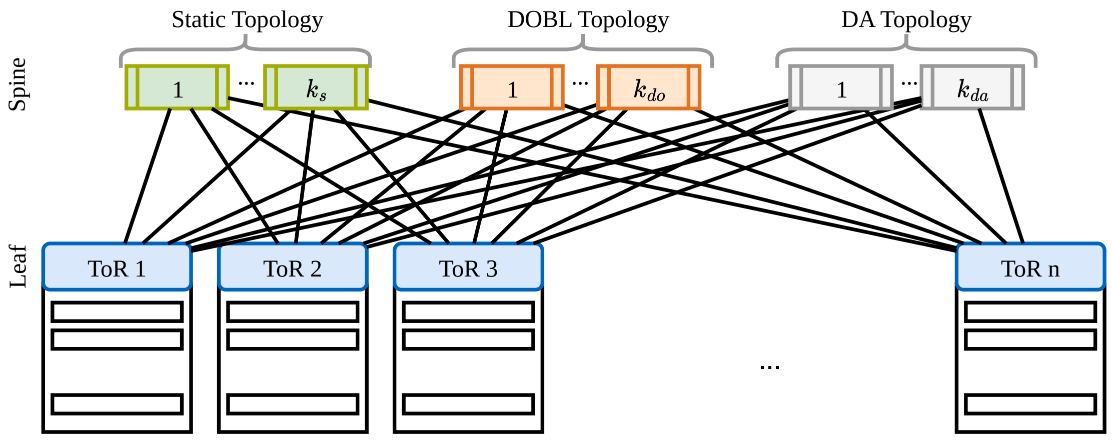


Figure 3.8 Schematic view of the ToR-Matching-ToR model for describing RDCNs.

gives the name of ToR-Matching-ToR (TMT) model. The TMT can model existing architectures such as RotorNet [48], Opera [49] or ProjecToR [47], by supporting multiple switch types.

Figure 3.8 illustrates a schematic view of the TMT model. It interconnects a set of n ToR (packet) switches, $N = \{t_1, t_2, \ldots t_n\}$, and a set of k spine (circuit) switches, $SW = \{sw_1, sw_2, \ldots, sw_k\}$. Internally, each switch connects its ingress- and egress-ports via a matching. Each ToR i $(1 \leq i \leq n)$ has k uplinks, where uplink $j : 1 \leq j \leq k$ connects to port i at (spine) switch sw_j. The uplinks can be separated into egress and ingress directions. The directed egress (leaf) uplink is connected to the ingress port of the (spine) switch, and the directed ingress (leaf) uplink is connected to the egress port of the (spine) switch. The matchings inside the spine switches are directed from ingress to egress The model is agnostic of the technological details of the optical link and can be used for any bandwidth (e.g., 10 Gbps as in [211, 48, 49]) and higher rates, e.g., 50 Gbps as in [53] or 100 Gbps. At any point in time and if it is not reconfiguring, each switch $sw \in SW$ *matches* its ingress and egress ports. The matching may be *reconfigured* at runtime to another matching. Each switch j has a set of one or more (feasible) matchings $\mathcal{M}_j$ where $m_j = |\mathcal{M}_j|$ is the number of matchings. Changing from a matching $M' \in \mathcal{M}$ to a matching $M'' \in \mathcal{M}$ takes time R_j: the *reconfiguration delay* of switch j. Furthermore, the model assumes that, during reconfiguration, the links in $M' \cap M''$, i.e., the links which are not being reconfigured, can still be used for forwarding (not-stop-all switches) [189]. Depending on the technology, different switches in SW support different sets of matchings and reconfiguration delays.

3.7 Summary

This chapter presents the technological background for reconfigurable topologies and introduces the two considered use cases. Using the components of a reconfigurable link, it elaborates on the three main characteristics we analyze in this thesis: reconfiguration delay, networking layers considered during reconfiguration, and classes of reconfiguration. The summary of the most prominent related work for both use cases provides a broad overview of the area's state-of-the-art. For RDCNs, the TMT model is presented to describe a significant body of the prior work. Moreover, the chapter hints at existing research gaps with respect to the characteristics. A more in-depth discussion of related work for the individual characteristics is left to the respective chapters.

Chapter 4

Measurements, Modeling, and Emulation of Topological Reconfigurability

Reconfiguring topologies comes at a cost: the time it needs to reconfigure the optical hardware. Meanwhile, data rates both in datacenter networks (DCNs) and in wide area networks (WANs) are increasing to hundreds of gigabits. Hence, even interruptions of hundreds of milliseconds can diminish the benefit of adapting networks when reconfiguring frequently.

Indeed, optical hardware can cycle through given configurations with reconfiguration delays in the order of microseconds [52, 48] or nanoseconds [53]. However, this specialized kind of hardware is not widely available. Moreover, due to its cycling speeds, it is also limited in its programmability. The schedules are fixed or can only be changed on a coarse timescale [48, 49, 53]. This limits the application of such hardware to demand-oblivious (*DO*) reconfigurable datacenter networks (RDCNs). As an alternative to specialized hardware, commercially off-the-shelf (COTS) optical circuit switches (OCSs) provide reconfigurations in order of milliseconds and are reconfigurable on-demand, such that a scheduling or matching algorithm can determine the next configuration based on the network state. Several prototypes based on COTS OCS were built, and their advantages were demonstrated, e.g., [45, 46, 55, 191].

In order to facilitate the demanded planning and to operate such RDCNs efficiently, a solid understanding of the reconfiguration behavior, its stability, predictability, and influence factors is necessary. Having access to a performance model of reconfiguration delays can open possibilities to consider specific reconfiguration costs and, thereby, reduce negative impacts. Thus, it can ultimately help to reap the benefits of adaptive networks. Moreover, while promising performance results have been demonstrated with various experimental studies, it is often challenging to experiment with these technologies. They are usually based on custom-built prototypes and rely on tailored hardware and software which is not publicly available.

This chapter addresses these two issues and sheds light on the first characteristic of reconfigurable topologies, the reconfiguration delay. First, it evaluates the reconfiguration behavior of programmable optical networks, i.e., networks consisting of programmable switches, programmable end-devices and programmable optical hardware. To this end, the chapter presents a meta-analysis of existing work on topological reconfigurations and draws the landscape of such measurements. Based on the structure of a reconfiguration command, it then elaborates on potential factors of variability for the reconfiguration delay, such as the number and frequency of reconfigurations. It contributes three measurement procedures tailored for two classes of devices: programmable switches and programmable network interface cards (NICs). The procedures leverage the potential of programmable COTS equipment, i.e., they do not require any specialized measurement equipment. Finally, the results of a measurement campaign provide insights into the behavior of six programmable COTS networking devices under optical circuit reconfiguration and verify the hypotheses about the influence factors. Thereby, we extract the significant components of end-to-end reconfiguration delay.

Second, this chapter presents a flexible framework for building reconfigurable networks, ExReC, that only relies on commercially available COTS hardware. The framework supports experimentation and reproducibility and can be configured in different ways, allowing the emulation of different RDCN architectures that follow the ToR-Matching-ToR (TMT) model presented in Section 3.6. In particular, ExReC can emulate hybrid topologies consisting of components from both classes of topological reconfigurability, *DO* and demand-aware (*DA*). ExReC uses emulation to reduce dependence on hardware that is either expensive or not available. Therefore, it uses an electronic packet switch (EPS) and label-based routing. Moreover, since today's programmable switches are usually unsuited for the packet scheduling logic required by Time Division Multiple Access (TDMA)-based *DO* topologies (e.g., RotorNet [48]), ExReC implements this logic on generic servers. This increases the flexibility for implementing forwarding logic. The chapter validates and demonstrates ExReC under various workloads, also considering a distributed machine learning (DML) training application.

Content and Outline: Section 4.1 overviews prior work with respect to modeling and measurements of reconfiguration delays as well as available prototype implementations and experimentation frameworks. Based on the components of an optical link and a reconfiguration command, Section 4.2 models the reconfiguration delay and hypothesizes possible factors of influence. Section 4.3 presents a measurement procedure and overviews the available testbed components. More details on the measurement framework for the individual scenarios are given in Section 4.4 and Section 4.5, along with the results and an assessment of the influence factors. Section 4.6 and Section 4.7 present and validate ExReC. Finally, Section 4.8 summarizes this chapter.

This chapter is based on two previous conference publications [11] (Section 4.1 - Section 4.5) and [12] (Section 4.1, Section 4.6, and Section 4.7). As a contribution to the research community, the code and the hardware specifications from ExReC have been made available at `https://github.com/tum-lkn/exrec`.

4.1 Related Work

The related work for this chapter revolves around two directions: the modeling and measurement of reconfiguration delay and the experimentation and emulation of topological reconfigurability.

4.1.1 Modeling and Measurements of Topological Reconfigurability

Related work on measurements of topological reconfigurations spans two domains. First, this section reviews existing measurements of end-to-end reconfiguration delays and compares them to the values used in more theory-oriented works that contain only simulations or analytical evaluations. Second, it provides a selection of performance modeling in the area of programmable networks.

Table 4.1 Overview of existing reconfiguration delay measurements: name, if circuit switching is programmable, if (ToR) switches are used for aggregation, the endpoints of the links, if only commercially available equipment is used, measured data plane reconfiguration delay, and if reconfiguration delays are drilled down.

	Name	On-demand	Connects via switch	Data plane device (**S**witch/**F**PGA/**N**IC)	Uses COTS OCS	Delays	Drills down
DCN	Mordia [52]	✗	✗	Myricom 10G-PCIE-8B (N)	✗	11.5 µs	✗
DCN	ReacToR [50]	✗	(✓)	HTG-V6HXT-100GIG-565 (F)	✗	11.5 µs	✗
DCN	RotorNet [48]	✗	✗	Myricom 10G NICs (N)	✗	150 µs	✗
DCN	Sirius [53]	✗	✗	Xilinx UltraScale VCU108 (F)	✗	3.84 ns	✓
DCN	Helios [45]	✓	✓	Fulcrum Monaco 24-port 10G (S)	✓	27 ms	✓
DCN	OSA/ Wave-Cube [46, 55]	✓	✗	some 1G NICs (N)	✓	9 ms	✗
DCN	Flat-tree [51]	✓	✓	Not specified	✓	2 – 2.5 s	✗
DCN	xWeaver [191]	✓	✓	Not specified	✓	300 ms	✗
DCN	MegaSwitch [54]	✓	✓	Broadcom Pronto-3922 (S)	✗	3 ms	✓
DCN	ProjecToR [47]	✓	✗	TI DLP Discovery 4100 (F)	✗	12 µs	✗
WAN	RADWAN [151]	✓	n/a	Not specified	✓	≈ 70s	✗
WAN	Hall et al. [131]	✓	n/a	Not specified	✓	570 ms	✗
	This thesis	n/a	✓/✗	multiple	✓	7.5 ms – 1.5 s	✓

4.1.1.1 Measurements of Reconfiguration Delay in Optically-switched DCNs

While there is no dedicated measurement study for circuit switching in RDCNs, different prototypes exist and are evaluated with respect to their reconfiguration delay. These measurement results can be classified into two groups: RDCNs that use (pre-)calculated schedules and RDCNs with programmable, on-demand reconfigurations.[1] Table 4.1 overviews the measurements and the involved network devices. The table provides insights into the class of reconfigurations, if the measurement considers a switch (top-of-rack (ToR))-based scenario, the used data plane device, if only commercially available OCSs are used, the achieved delay, and if the measurement study elaborates on different components of the reconfiguration delay.

RDCNs with pre-calculated schedules. Mordia [52] proposes a custom-built OCS that supports arbitrary reconfigurations. The design features a 2D-microelectromechanical system (MEMS) and wavelength selective switches (WSSs) instead of 3D-MEMS. Thereby, Mordia achieves reconfiguration delays of 11.5 µs on average. The prototype emulates ToRs with generic servers, which are equipped with Myricom 10G-PCIE-8B dual port NICs and Dense Wavelength Division Multiplexing (DWDM) transceivers. A Field Programmable Gate Array (FPGA)-based controller communicates the active circuits and synchronizes the hosts via an out-of-band channel. The analysis does not provide a decomposition of the reconfiguration delays. ReacToR [50] extends the Mordia OCS prototype and emulates ToRs with FPGAs (HTG-V6HXT-100GIG-565). The reconfiguration delays are in line with those from Mordia. Also, ReacToR does not give a deeper analysis of reconfiguration delay or potential influence factors.

RotorNet [48] is another custom circuit switch implementation. Similarly to the previous ones, it provides host-based measurements of end-to-end reconfiguration delays using Myricom 10G NICs. The measurements show delays of 150 µs. Out-of-band notifications synchronize the hosts and the circuit switch. RotorNet follows a *DO* reconfiguration approach, i.e., reconfigurations are

[1]Note that this classification is slightly different to the reconfiguration classes presented in the literature overview in Section 3.5 since we are interested in the switching mode.

pre-determined and cannot be varied at runtime. Sirius [53] uses an optical fabric with passive components. Circuits are set up using arrayed-waveguide gratings and transceivers with tunable lasers that tune their wavelengths according to a pre-determined schedule. In the prototype, the endpoints of the links are realized via FPGAs (Xilinx UltraScale VCU108). Tuning the lasers can operate at nanoseconds speed so that end-to-end reconfiguration delays of 3.84 ns are achieved.

The aforementioned prototypes provide all end-to-end reconfigurations < 1ms. However, they address a different use case in which circuit scheduling is not performed on-demand but uses pre-defined schedules. Those schedules are either fixed or changed on a coarse time level.

RDCNs with on-demand reconfigurations. The majority of programmable RDCNs with on-demand reconfigurations uses full crossbar, 3D-MEMS-based OCS or Wavelength Division Multiplexing (WDM) switching. Helios [45] uses a commercially available Glimmerglass OCS with 64 ports and connects Fulcrum Monaco switches with 10G ports. The authors report on several configuration changes and firmware modifications necessary on the switch, such as deactivating debouncing and electronic dispersion compensation (EDC), which were conducted with support from the manufacturer. They drill down the reconfiguration time into two components: The reconfiguration time on the data plane is 27 ms; the control plane adds another 170 ms in case of synchronous requests to the OCS and 30 ms in case of asynchronous requests. However, the measurements do not investigate how reconfiguration parameters or networking devices affect end-to-end reconfiguration delay.

OSA [46] and WaveCube [55] use the same testbed. It consists of a Polatis Series 1000 32-port OCS and a CoADna WSS. The measurements of the optical receive power report reconfiguration delays of 14 ms for the WSSs and 9 ms for the OCS. The authors do not provide further insights into the end-to-end reconfiguration delays nor details about the used NICs that connect to the optical fabric. Flat-tree [51] uses a commercially available 192-port OCS. Xia et al. estimate the end-to-end reconfiguration time based on observed TCP throughput between $2 - 2.5$ s. This also includes forwarding rule updates on the ToR switch. The authors do not report how individual components contribute to the end-to-end delay. Similarly, coarse observations are provided in xWeaver [191]. Here, the observed time for the TCP throughput to restore after reconfiguration is around 300 ms. Again, the authors do not provide details of the used equipment.

In contrast, MegaSwitch [54] provides a detailed overview of the used devices. The prototype builds around Broadcom Pronto-3922 switches, running in OpenVSwitch mode, with InnoLight 10GBASE-ER DWDM SFP+ transceivers. The custom-built optical fabric uses WSSs and is controlled by a Raspberry Pi. The authors split the total reconfiguration time into three components: the optical reconfiguration delay (3 ms), the added overhead due to flow updates on the packet switches (13 ms), and the control plane delay (7 ms). While this provides an initial model for the total reconfiguration time, it does not cover potential influence factors of the individual components.

ProjecToR [47] relies on free-space optics for switching and uses TI DLP Discovery 4100 kits with 0.7 XGA chipsets as forwarding equipment. The reconfiguration measurements focus on the loss of light duration due to mirror adjustments and are in the order of 12 µs. Measurements of end-to-end reconfiguration delays are not provided. Due to the distributed nature of ProjecToR, it differs from our use case.

In conclusion, existing measurements for RDCNs with on-demand reconfigurations that rely on commercially available equipment consistently achieve reconfiguration delays in the order of milliseconds. The modeling and evaluation of impact factors is basic, and only individual points are provided. A comprehensive investigation and model are missing.

4.1.1.2 Measurements of Reconfiguration delay in Demand-aware WANs

Reconfiguration delays in WANs have not been studied extensively. One exception is RADWAN [151], which provides a brief measurement of the reconfiguration delay of the modulation format in a lab environment. The study gives a singular delay of ≈ 70s. It does not vary any parameters, and the detailed settings are not specified. Hall et al. [131] evaluate the reconfiguration delay of band multiplexing modules. Again the measurement considers a lab environment. Specifically, they compare the automated tuning of link parameters with the (manual) restoration of previously cached configuration parameters for the amplifiers. The achieved reconfiguration delay for new demand situations is ≈ 20s, whereas the restoration of cached values takes only ≈ 0.57s. The measurement revolves around optical performance indicators but does not show end-to-end reconfiguration delay. Moreover, the exact equipment used is not specified.

4.1.1.3 Reconfiguration Delays Used in Theory-oriented Studies

Prior algorithmic or theory-oriented studies orient at measurement results such as those listed above. Depending on the particular approach targeted by the study, the reconfiguration delays vary. Works that aim at pre-calculating schedules consider the micro-second scale values. For instance, Solstice [187] uses 20 µs; Sunflow [189] considers delays between 10 µs and 100 ms. Similarly, Vargaftik et al. [128] use 20 µs and 20 ms depending on the evaluated switch. C-Share represents an approach that calculates circuits on demand. The used delay is 20 ms. In summary, these works use measurement results of other works and do not consider the potential variability of the reconfiguration delay in their evaluations.

4.1.1.4 Performance Models for Programmable Network Devices

Predictability and evaluation of performance models as a general concept have already been explored for other aspects of programmable networks. Harkous et al. [212] analyze the influence of P4 constructs on packet processing latency. Scholz et al. [213] model packet rates, latencies, and resource consumption of two classes of P4 targets in dependence on matching types and table sizes of P4 programs. Katsikas et al. [214] provide such evaluations for programmable NICs and further evaluate the impacts of reconfiguring such devices. Performance modeling of programmable control planes has been conducted too. For instance, van Bemten et al. [215] assess the predictability of commercially available OpenFlow switches with respect to the correctness and latency of reconfigurations. Other works provide specific benchmarking and modeling tools for such software defined networkings (SDNs) [216, 217].

4.1.2 Experimentation & Emulation

Also regarding experimentation and emulation of topological reconfigurability, prior work can be discriminated according to the reconfiguration class. Table 4.2 summarizes existing experimental evaluations of the two classes, the scales, and the availability of the components from a hardware and software perspective.

Table 4.2 Overview of experimental platforms for reconfigurable networks: name, basic methodology, achieved scale, link rate, if DO is available, if DA is available, if only COTS equipment is used, and if artifacts to reproduce measurements are available.

Name	Basis	Scale	Link rate	DO	DA	COTS	Source Code
RotorNet [48]	HW	8 emulated ToRs	10 Gbps	✓Prototype	✗	✗	n/a
Opera [49]	HW	8 emulated ToRs	10 Gbps	✓emulated (P4)	✗	✓	n/a
Sirius [53]	HW	4 nodes	50 Gbps	✓	✗	✗	n/a
Etalon [204]	Emulation	8 emulated ToRs (128 hosts)	10 Gbps	✓emulated (Click Router)	✗	✓	✓
Helios [45]	HW	24 End-hosts	10 Gbps	✗	✓	✓	n/a
xWeaver[191]	HW	10 emulated ToRs	10 Gbps	✗	✓	✓	n/a
OSA/WaveCube [46, 55]	HW	8 emulated ToRs	unclear	✗	✓	✓	n/a
Flat-Tree [51]	HW	24 End-hosts	10 Gbps	✗	✓	✓	n/a
ProjecToR [47]	HW	3 ToRs	-	✗	✓Prototype	✗	n/a
MegaSwitch [54]	HW	40 End-hosts	10 Gbps	✗	✓Prototype	✗	n/a
Mordia/ReacToR [52, 50]	HW	23 End-hosts	10 Gbps	✗	✓Prototype	✗	n/a
c-Through [56]	HW	4 emulated ToRs, 16 hosts	1 Gbps	✗	✓emulated (host-based)	✓	n/a
C-Share [182]	Mininet	10 leaf switches	100 Mbps	✗	✓emulated (OpenVSwitch)	✓	n/a
PSNet [103]	HW	3 ToRs	1 Gbps	✗	✓emulated (OpenVSwitch)	✓	n/a
This thesis	HW	8 emulated ToRs	10 Gbps	✓emulated (DPDK, VLAN)	✓	✓	✓

4.1.2.1 Demand-oblivious RDCNs

As already observed in Section 4.1.1, *DO* switches are not commercially available. Thus, all experimental assessments rely on prototype implementations. For instance, the seminal work RotorNet [48] uses the rotor switch implementation presented in [120]. It emulates ToRs on physical servers and connects them via 10 Gbps links. In contrast, Opera [49] is evaluated with a P4 switch-based emulation using only COTS components. Specifically, it uses an 6.5-Tbps Intel Tofino which emulates all ToR switches as well as the circuit switching fabric. Although they authors rely on COTS components only, the framework is not publicly available. The Sirius prototype [53] uses FPGAs as endpoints of the links. The FPGAs control the custom-built tunable transceivers in order to cycle through the matching. The setup achieves a link rate of 50 Gbps. Etalon [204] provides an emulation of *DO* topologies using only COTS components. It emulates multiple ToRs (and racks) per physical server in the testbed and uses time dilation to achieve higher rates. However, it only relies on software-based forwarding solutions and fully virtualizes the topology. The framework is available but only considers a *DO* topology so far.

4.1.2.2 Demand-aware RDCNs

DA switches can be configured to create direct connections between ToRs according to the traffic conditions. They have a high degree of freedom and can realize any possible matching. When a circuit is established, it provides constant datarate between ToRs. As observed in Section 4.1.1, the reconfiguration delay of *DA* switches r_{da} is usually larger than the reconfiguration delay of *DO* switches r_{do}.

For *DA* reconfigurable topologies, a wider range of experimental evaluations has been conducted. A group of studies relies on commercially available OCSs, e.g., Helios [45], OSA/WaveCube [46, 55], xWeaver [191] or Flat-Tree [51]. They present setups that represent ToRs by physical servers. The considered sizes are in the order of tens of end-hosts and they support link rates of 10 Gbps. However, for none of them, the source code is available. Other studies explore different types of interconnects, e.g., ProjecToR [47], MegaSwitch [54] or Mordia [52], but use prototype implementations of the switches that are not available. Finally, *DA* topologies have also been evaluated using emulations. For instance, c-Through [56] considers host-based scheduling, i.e., the hosts know the active circuits and only send traffic accordingly. C-Share [182] uses Mininet [218] and OpenVSwitch for OCS emulation. Specifically, the OCS is emulated by an OpenVSwitch that forwards packets on an input port only to one output port at a time. The achieved link rates (100 Mbps) are bounded by the performance of the server. Finally, PSNet [103] is another example for a testbed implementation of a *DA* RDCN which relies on OpenVSwitch to emulate circuit switching. The setup is rather small with 3 ToRs and 6 end-hosts and also the source code is not available.

Summary

The state of the art provides only singular measurements of reconfiguration delays for COTS devices. It has not been thoroughly explored if and how existing programmable networking equipment can cope with such reconfiguration. In particular, a benchmark across different devices and varying reconfiguration scenarios has not been done yet. This is critical as previous evaluations of programmable networking devices showed that they behave differently, e.g., with respect to OpenFlow updates [215]. Moreover, the discussed theory-oriented works make varying assumptions about reconfiguration delays and often base them on previous measurements or the data sheets of the OCSs [187, 128, 189]. Yet, these figures do not always consider the end-to-end reconfiguration delay of optical links and the delay on the control plane. In addition, they neglect potential variability, i.e., rely on the assumption that end-to-end reconfigurations are deterministic. The review of programmable, on-demand RDCN prototypes shows consistent values for the reconfiguration delays. But the end-to-end reconfiguration delay of programmable optical links has not been modeled in detail yet.

Finally, all the experimental evaluations of reconfigurable topologies consider either *DO* or *DA* switches but not a hybrid setting. In addition, in all but one case, the frameworks are not available.

4.2 Modeling End-to-end Reconfiguration Delay of Programmable Optical Links

Figure 4.1 shows a programmable optical link. The link is terminated by control- and/or data-plane programmable interfaces, e.g., an end-host-based (Smart-)NIC or FPGA (left), or an OpenFlow or P4-capable switch (right). The NICs are equipped with duplex optical transceivers to send and receive data via separate circuits. The optical link traverses an on-demand configurable (programmable) OCS. An external controller handles the components.

A reconfiguration takes several steps. First, the controller sends the new configuration to the OCS. After processing the command, the OCS sets the new circuit in place. The transmitter and

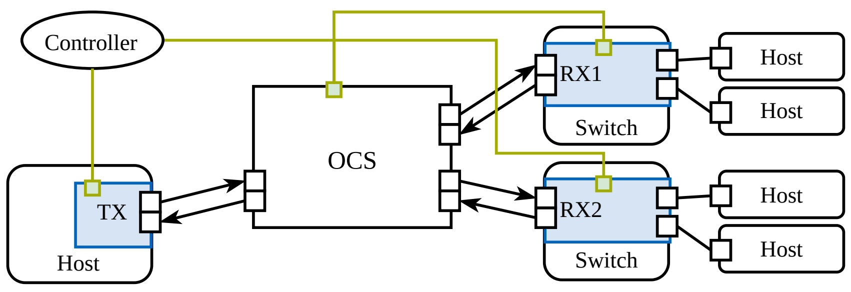


Figure 4.1 Example of a programmable optical link. Three programmable networking devices (1 host, 2 switches) are connected via an on-demand reconfigurable OCS. An external controller handles all the components.

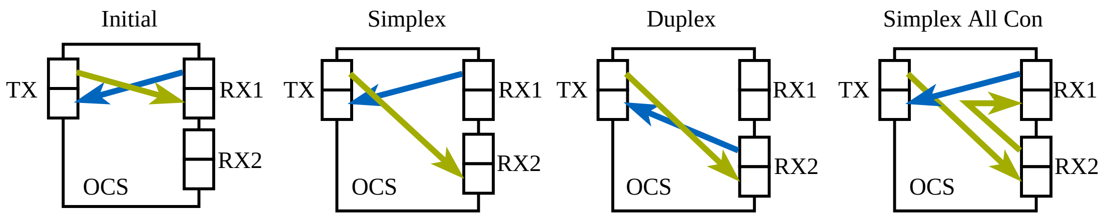


Figure 4.2 Variants of circuits and reconfigurations on the OCS. Ports `TX`, `RX1`, and `RX2` connect to programmable networking devices. Each port consists of two fibers (in/out). The figure left (Initial) shows the initial circuits. The other three parts show the reconfiguration cases: *Simplex*, *Duplex* and *Simplex All Con*.

receiver of the new circuit need to synchronize their clocks so that the receiver can correctly recover the signal [53]. Then, they communicate link connectivity to higher layers and ultimately to the (forwarding) application. Eventually, the controller updates forwarding rules or configuration of the programmable devices. The exact process depends on the specific firmware and operating system of the device.

This study focuses on the optical switching part. Reconfiguration delays due to forwarding rule updates are out of scope here. Similarly to previous proposals [45, 54], the model contains two components for the end-to-end reconfiguration delay: data plane downtime r_d and control plane delay r_c. Since processing on the control plane happens before the data plane is interrupted, the sum of both provides the total delay:

$$r = r_d + r_c. \tag{4.1}$$

The reconfigurations can vary in several dimensions with potential impacts on the reconfiguration delay. Influence factors can be all attributes of a reconfiguration command that the controller can modify. Candidates are the number of modified ports n and the distances d between the ports, which are reconfigured.

Moreover, the data plane downtime might depend on the used devices x (categorical variable). The review of related work has shown that two classes of devices are being used: host-based and switch-based interfaces (cf. Figure 4.1 left and right). Many proposed designs conceptually rely on a (programmable) ToR switch as an aggregation point towards the circuit-switched fabric, e.g., [45, 54, 191]. Interestingly, directly connecting an OCS to the end-hosts can provide benefits in certain situations, e.g., for DML [219].

Since the deployed transceivers are bidirectional, i.e., use two circuits, there are three ways of reconfiguration: *Simplex* (SI), *Duplex* (DU), and *Simplex All Con.* (SAC) (Figure 4.2). The ports `TX`, `RX1` and `RX2` connect to bidirectional optical transceivers, e.g., as shown in Figure 4.1. SI changes only the circuit, which carries the traffic to `RX2`. This represents the case of setting unidirectional links, a widely adopted model in theoretic work [187]. Moreover, it is the least demanding reconfiguration at the OCS as it requires only one change. Thereby, `TX` always receives optical power on its receiver, also during the reconfiguration. Consequently, link establishment or failure routines defined by IEEE 802.3 are not triggered on `TX`. However, since `RX1` no longer receives light, some NICs might run these routines at `RX1`, leading to unintended behavior. For instance, shutting off the transmitter of `RX1` results in a loss of signal at `TX` which in turn stops transmitting [45]. To avoid such disconnects, DU configures both circuits to `RX2`. This is also a natural choice if traffic is transmitted bidirectionally. Finally, SAC constructs a ring over all involved transceivers with the same intention as DU, i.e., to avoid triggering link failure routines. It creates unidirectional circuits and, thereby, provides more flexibility [45]. The reconfiguration type c is the fourth impact factor.

In summary, we hypothesize as relation for the data plane downtime:

$$r_d = D(n, d, x, c). \tag{4.2}$$

The control plane delay might also depend on the number of modified ports n. Moreover, COTS OCSs usually come with a number of control plane protocols p (categorical), e.g., TL1 [220] or Netconf [221]. We obtain for the control plane delay:

$$r_c = C(n, p). \tag{4.3}$$

4.3 Measuring End-to-end Reconfiguration Delay of Programmable Optical Links

This section describes the procedures as well as the testbed components for measurements to evaluate the hypothesized relationships above.

4.3.1 Data plane measurements

To assess the hypothesized relations for the data plane downtime, a single measurement follows the process shown in Figure 4.3. A traffic source sends a continuous stream of packets over an existing optical link from `TX` to `RX1`. The packet stream samples the state of the OCS, i.e., the port that packets are received on (`RX1` or `RX2`) directly shows the established circuit. The controller sends a command to reconfigure the circuit to another receiving port (`RX2`) at time t_{cmd}. The data plane downtime is given as

$$r_d = t_{\mathtt{RX2}} - t_{\mathtt{RX1}} \tag{4.4}$$

where $t_{\mathtt{RX1}}$ is the time of the last packet received by `RX1` and $t_{\mathtt{RX2}}$ is the time of the first packet received by `RX2`.

4.3.2 Control plane measurement

The measurement of the control plane delay uses a similar approach as the data plane downtime one. It excludes any timings needed to calculate the new configuration, e.g., for collecting demand

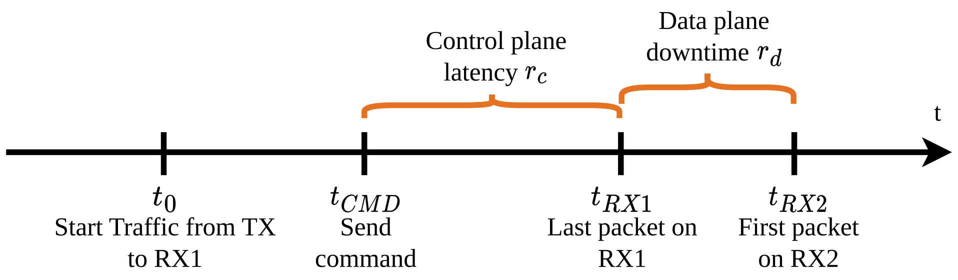

Control plane latency r_c
Data plane downtime r_d
t
t_0 Start Traffic from TX to RX1
t_{CMD} Send command
t_{RX1} Last packet on RX1
t_{RX2} First packet on RX2

Figure 4.3 Conceptual timeline of the reconfiguration delay measurement. Characteristic duration of reconfiguration (red braces). `RX1` and `RX2` are the ports between the circuit is changed.

statistics or executing optimization algorithms. Accordingly, r_c is defined, as shown in Figure 4.3, from sending the command message $t_{\mathtt{CMD}}$ to the OCS until reception of the last packet on `RX1`:

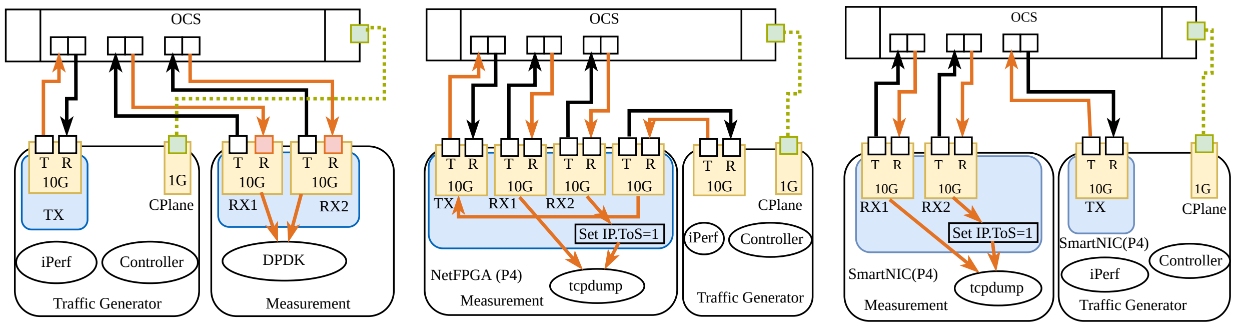
$$r_c = t_{\mathtt{RX1}} - t_{\mathtt{CMD}}. \tag{4.5}$$

4.3.3 Testbed

The testbed is built around a Polatis Series 6000n OCS [222] with 32 in- and 32 out-ports. The OCS runs firmware version 6.7.5.34. All evaluated devices use FS.com 10GBASE-LR SFP+ 1310nm (SMF) transceivers [223]. The measurements compare two classes of data plane devices: programmable NICs deployed in hosts and programmable switches. For the host-based cases, there are three NICs with different expected levels of programmability (from low to high): a regular NIC (Intel X710-DA4 [224]) with Data Plane Development Kit (DPDK) [225] support, a SmartNIC (Agilio CX 2x10G [226]) with support for P4 and DPDK and an FPGA (NetFPGA-SUME [227]) with prepared support for P4 and the full programmability of FPGAs.

The switch-based measurements are conducted using two OpenFlow-based switches, Pica P3297 [228] and Dell S4048-ON [229], and one Layer 2 switch, EdgeCore AS5835-54X [230], running SONiC [231]. The switches run in the default configuration and only make simple forwarding decisions. The exact settings of the measurement cases are detailed later (Section 4.4.1.1 for host-based and Section 4.4.1.2 for switch-based).

Traffic and packet dumps are generated on two servers running Ubuntu 18.04 (4.15.0-47-generic kernel) with 128 GB of RAM and an Intel Xeon Silver 4114@2.2 GHz (20 cores). The traffic-generating server also runs the controller of the OCS, which connects via a dedicated management network. It uses iPerf 2.0.10 [232] to generate a single UDP flow consisting of 1000 B packets at a rate of 2 Gbps ($\approx$ 250 Kpps). This translates to a packet inter-arrival time of $\approx$ 4 µs and determines the lower bound of the measurement range of the reconfiguration delay. Since the reconfigurations are expected to be in the order of milliseconds, this provides a sufficient temporal resolution.[2] The presented results contain 30 runs with traffic running at least 2 s before and after the reconfiguration. The OCS controller is implemented in Python and uses the available programming interfaces of the OCS. If not stated otherwise, it uses the Netconf interface.



(a) DPDK-based. **(b)** P4-based with NetFPGA. **(c)** P4-based with SmartNIC.

Figure 4.4 Testbed setups of the host-based measurements: Setups consist of a traffic generator, which includes also the OCS controller, and a measurement server. The measurement server hosts the NICs under test. Arrowheads indicate the direction of optical light and packets. Red arrows indicate the packet streams that are to be measured.

4.4 Evaluating Data Plane Downtime

We start with evaluating the data plane component. After introducing the detailed configurations, results for the data plane devices are compared, and potential influence factors from Section 4.2 are analyzed.

4.4.1 Detailed Setups

The settings to measure data plane reconfiguration behavior in the host-based and switch-based cases vary slightly due to the capabilities of the considered devices. The general assumption is that the links are homogeneous, i.e., that both endpoints use the same device model. Combining different devices makes attributing effects harder and is left for future work. Overall, this leads to three setups:

4.4.1.1 Host-based scenarios

The two data plane programming abstractions DPDK and P4 provide sufficient precision for time measurements in the order of milliseconds [233, 234]. However, neither DPDK nor P4 is supported by all the devices under test. Hence, this section proposes two approaches and evaluates the impact of using DPDK or P4 on the measurements. The SmartNIC supports both technologies, which helps to put the obtained results into relation.

DPDK The first approach (Figure 4.4a) uses a custom DPDK application to measure the data plane downtime. The application consists of two threads. Each thread polls bursts of packets from its assigned port, i.e., `RX1` or `RX2`. When the application receives packets, it reads the current cycle count (from the central processing unit (CPU)) and saves it as the last packet reception time. Similarly, the application stores the timestamp of the first received packet per port. The data plane downtime is directly calculated from these collected timestamps. The DPDK application uses the available poll mode driver of the NICs. It is available for the Intel NIC and the SmartNIC.

[2]Specifically, the data sheet of the OCS states a switching delay of 25 ms.

However, it relies on software timestamping since the used Intel NIC does not support hardware timestamping.

P4 The second approach uses P4 as the mean of programmability. Thus, it is limited to the NetFPGA and the SmartNIC. A simple P4 application forwards packets from the traffic generator onto the dynamic link. The setups differ between these two due to the different number of ports of the cards. The setup with the NetFPGA leverages all four ports of the device (Figure 4.4b). The traffic generator connects via one of the physical ports, and the NetFPGA forwards the traffic to the OCS. For the SmartNIC (Figure 4.4c), the traffic generator runs on the end-host that is one of the link endpoint and directly connects to the OCS. In both cases, packets received on either `RX1` or `RX2` are forwarded to a virtual port that is exposed to the operating system. Since this setup hides the link reconfiguration from the operating system (all traffic is received on the single virtual port), the packets are marked with the information of the receive port. Specifically, the P4 application sets the IP type of service field (ToS) to 1 when the packet is received on `RX2`. In addition, the application adds an in-band network telemetry (INT) header containing the time of the packet reception. The precision of the timestamp is 5 ns on the NetFPGA[3] and 20 ns on the SmartNIC[4]. Finally, the packets are collected using *tcpdump*.

This setup provides two ways to obtain the downtime: from the timestamps in the INT data and from the timestamps in the packet trace. The downtime is given as the difference in the timestamp of the first packet with IP.ToS = 1 and the timestamp of the last packet with IP.ToS = 0. If not stated otherwise, we use the INT data.

4.4.1.2 Switch-based scenarios

Figure 4.5 shows the setup for the switch-based scenarios. It follows a similar idea to that of the P4-based measurements with NetFPGA. The traffic generator sends traffic to the switch under test. The switch forwards the packets without modification on another port (`TX`) to the OCS. The OCS loops back to a third port (`RX1`) or fourth port (`RX2`). For packets received on `RX2`, the switch sets IP.ToS = 1. Independent of the receive port, packets are forwarded to the initial port that connects to the traffic generator. The outgoing direction of this port connects to a fiber tap and eventually to an Endace DAG 10X4-S measurement card to dump the packets. P3297 and S4048-ON implement this forwarding using OpenFlow rules, whereas the AS5835-54X uses a Virtual LAN (VLAN) header to indicate the receive port.

4.4.2 Comparing devices and reconfiguration cases

This section first presents the results for the host-based settings followed by the switch-based ones.

4.4.2.1 Host-based

Figure 4.6a illustrates the data plane downtime for the host-based devices and the different measurement approaches. It further compares the three reconfiguration cases SI, SAC, DU,

[3]The value was retrieved from `https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview#p4-netfpga-extern-library`.

[4]This value was received via correspondence with Netronome.

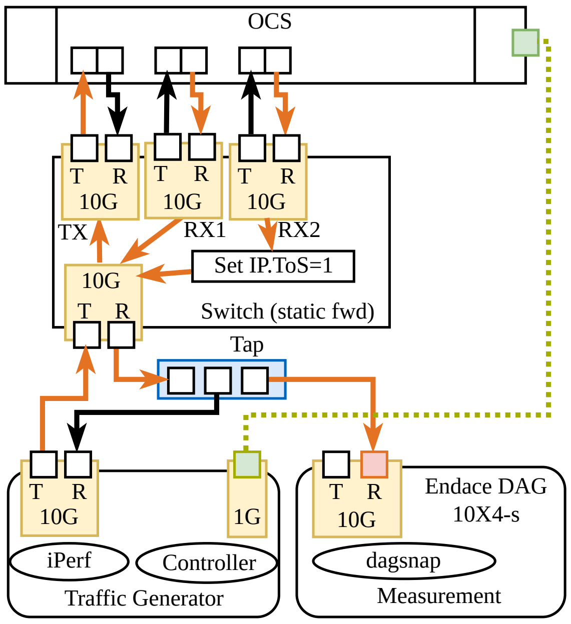


Figure 4.5 Testbed setup for the switch-based measurements: It consists of the switch under test, a traffic generator, which includes also the OCS controller, and a measurement server. Arrowheads indicate the direction of optical light and packets. Red arrows indicate the packet stream that is dumped by an Endace DAG 10X4-S.

which results in 12 settings. The figure shows violinplots across the 30 runs. Purple circles indicate the median values. We first observe that only SmartNIC provides consistent values of ≈ 7.5 ms for all three reconfiguration settings and also has low variance across the measurement runs. If only one circuit is modified (SI), the Intel NIC loses connectivity, i.e., no packets are received on `RX2`. Hence, the downtime cannot be calculated. SmartNIC and NetFPGA have median downtimes in the order of 10 ms. However, NetFPGA's behavior is hardly predictable. It shows high variability with values > 100 ms.

In the DU case, all candidates manage the reconfiguration. SmartNIC achieves around 7.6 ms, while NetFPGA is around 11.8 ms. For SmartNIC, the difference between the INT-based and DPDK-based results is about 0.5 ms on average. Thus, both measurement approaches are viable here. For the Intel NIC, the average and median downtime is around 292 ms. This is significantly larger than the expected 25 ms of the OCS and compared to the other devices. An explanation might be given by the specific firmware or driver, e.g., with slow implementations of link setup routines.[5]

4.4.2.2 Switch-based

Figure 4.6b shows violinplots of the data plane downtimes for the switch-based measurements. Again, the purple circles indicate the median values. We note that the SI case does not succeed for any switch. The switches turn off `RX1` and subsequently also `TX` so that no more packets are transmitted after the reconfiguration has finished. For SAC and DU, P3297 and S4048-ON show a data plane downtime > 1 s on average and in the median. While for P3297 both timings are around ≈ 1.1 s, S4048-ON has larger downtimes for DU (≈ 1.5 s) than SAC (≈ 1.2 s). AS5835-54X

[5]For verification, the measurements were repeated with a different firmware version and Intel was contacted to obtain explanations about this behavior, however without any success.

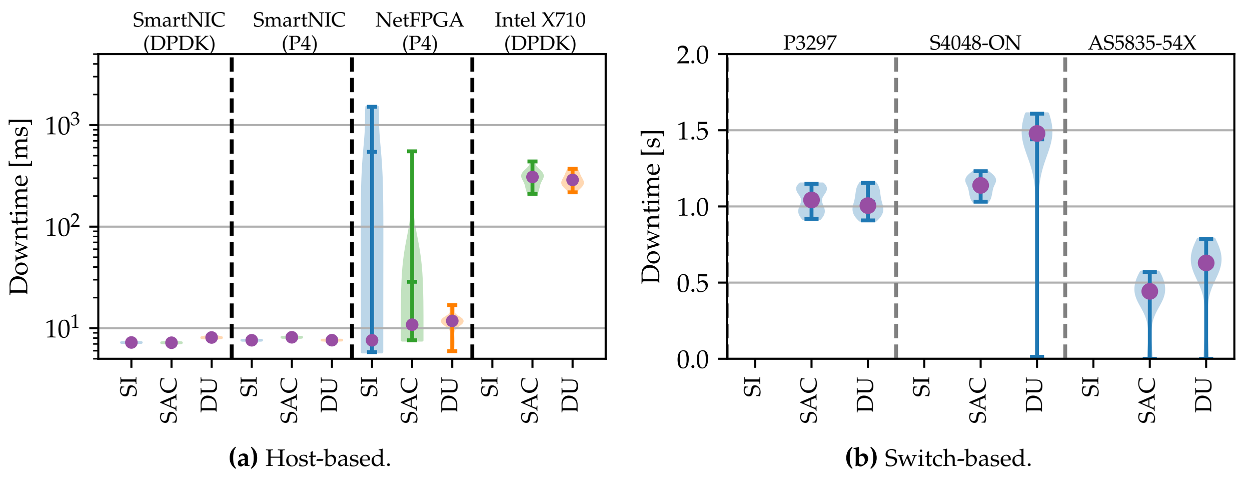


Figure 4.6 Violinplots of the data plane downtime. Comparison of the six devices, the reconfiguration cases (SI,DU,SAC), and measurement approaches (P4, DPDK). Only SmartNIC shows low downtimes of ≈ 7.5 ms and low variance. All switches show downtimes > 0.4 s. SI does not succeed for the switches and the Intel NIC (no values).

has smaller data plane downtimes of 0.442 s for SAC and 0.629 s for DU in the median and on average. Also, here, we observe the prolonged interruption for the DU case. The behavior is consistent over all runs, but the reasons could not be clarified.

The conclusions for r_d in the switch-based scenarios are negative. Data plane downtimes above 400 ms translate to either large reconfiguration periods or extreme losses of network capacity over time, hindering COTS packet switches on the edge of a circuit-switched or hybrid fabric. These observations confirm the ones from previous work [45, 54]. As suggested by [45] and also confirmed in discussions with two industry partners, this behavior is not related to the switching ASIC but roots in the firmware of PHY and MAC layer components, e.g., stems from the tuning of physical link parameters during link setup.

4.4.3 Analyzing Influence Factors

After evaluating the reconfiguration behavior of the six networking devices, this section focuses on how the number of modified circuits and the port distance impact the data plane downtime. We use the SmartNIC for the following analysis since it performed best in the previous measurements.

4.4.3.1 Number of modified circuits

Figure 4.7 shows violinplots of the data plane downtime against the number of modified circuits. The green circles indicate the median. The number of circuits ranges up to 32, which marks the maximum number of circuits that can be set simultaneously with a 32x32 OCS. The values of data plane downtime range from 7.5 ms to 8 ms. Moreover, the median value decreases slightly with the number of modified circuits. Fitting the following linear model

$$D(n) = \delta + \gamma \cdot n \tag{4.6}$$

shows a slope of $\gamma = -8.39 \cdot 10^{-7}$ and an intercept of $\delta = 0.0076$s with $p = 0.0032$, which is significant for $\alpha = 0.05$. However, the slope value is below the temporal resolution of the measurement setup, which is ≈ 4 µs according to the packet sending rate. Thus, this decrease has

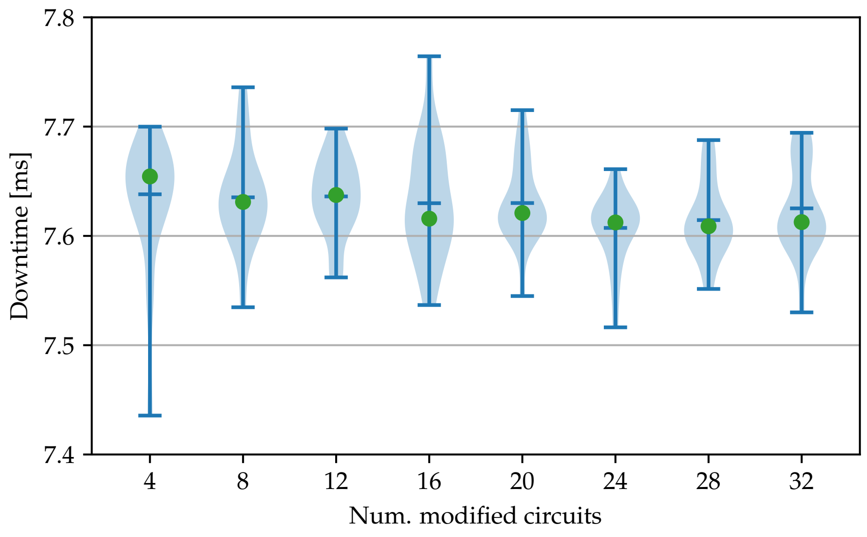


Figure 4.7 Violinplots of the data plane downtime against the number of modified circuits. The setup uses SmartNIC (P4), SI case. No significant impact is observable.

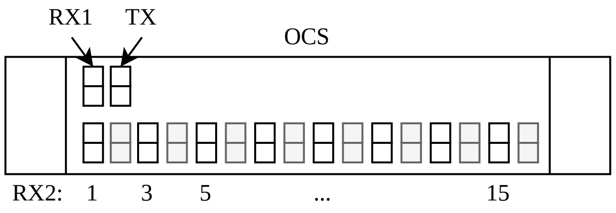


Figure 4.8 Schematic view of the definition of the port distance. The distance is defined using the physical embodiment of the OCS. The distance is given by the indicated numbers for `RX2`.

to be considered as measurement noise, and the conclusion is that there is no significant impact of the number of modified circuits on the data plane downtime of a single link.

4.4.3.2 Port Distance

A second parameter that can impact the downtime is the (physical) distance between the ports `RX1` and `RX2` at the OCS. Intuitively, one would assume a linear relationship between distance d and the data plane downtime:

$$D(d) = \delta + \gamma \cdot d. \tag{4.7}$$

Figure 4.8 indicates `RX2` with increasing distance to `RX1` (1, 3, 5...). The distances are given as the number of ports between `RX1` and `RX2` on the horizontal axis from an outside view of the OCS.

The results (Figure 4.9) show no clear pattern. The major part of the values lies between 7 and 8 ms. For most configurations, the variance is low (< 0.01 ms). Exceptions are distances of 7 and 11 ports, for which some outliers exist. Fitting the model gives a slope of $\gamma = 6.81 \cdot 10^{-6}$ with $p = 0.273$, which is not significant. Thus, we cannot conclude if the port distance linearly affects the data plane downtime. A deeper analysis, also considering the internal structure of the OCS, is left for future work.

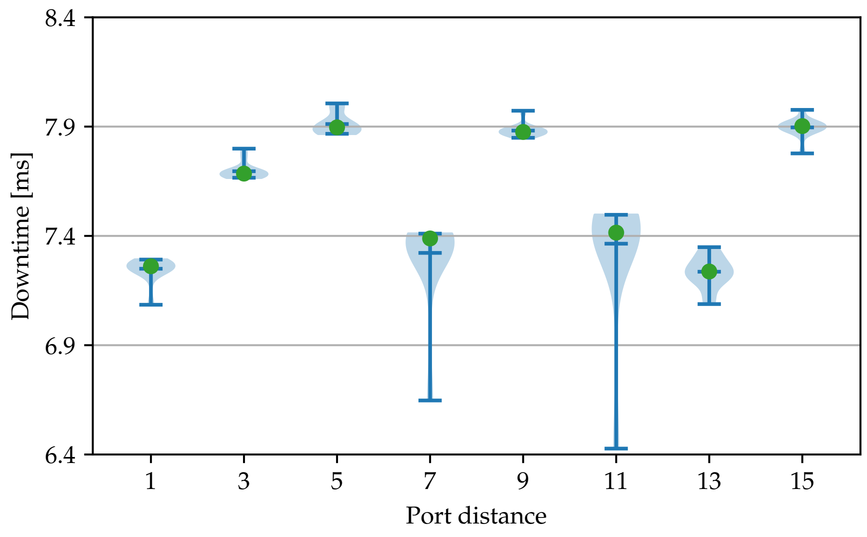
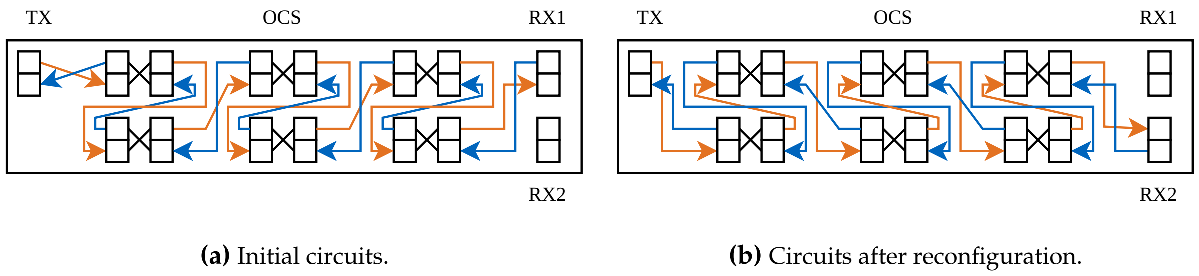


Figure 4.9 Violinplots of the data plane downtime against the port distance. Downtime is measured with SmartNIC (P4), SI case. Port distance is viewed from an outside perspective. No clear pattern is visible.



(a) Initial circuits. **(b)** Circuits after reconfiguration.

Figure 4.10 Circuits and fibers configured during the shoe-lace measurement. Black lines indicate fibers. Colored arrows are circuits. The optical link uses all ports of the OCS. The figure shows a sub-set of ports due to readability.

4.4.3.3 Chaining reconfigurations

Besides traversing a single OCS, an optical link might also span multiple OCSs. This is not only a common situation in WANs but might also become more common in DCNs, e.g., when scaling to larger topologies. The sizes of COTS OCSs are currently limited to several hundred ports which may in turn lead to multi OCS-layer network designs.

In order to assess the reconfiguration behavior in such a scenario, a setup similar to the shoelace proposed in [215] is used. Figure 4.10 illustrates the circuits before and after reconfiguration. The other components are the same as for the SmartNIC setup in Figure 4.4c. `TX` is connected to the traffic generator, and `RX1` and `RX2` are connected to the respective ports of the SmartNIC on the measurement host. The idea is to redirect the signal through multiple ports of the OCS with fiber loopbacks. The controller requests to reconfigure all these ports simultaneously with a single command. This represents the centrally coordinated reconfiguration of such a link. Specifically, this kind of measurement provides the time between the first port being disconnected and the last port being connected.

Figure 4.11 shows violinplots for the data plane downtime against the number of chained circuits, i.e., the length of the shoelace. In order to rule out an increase of the reconfiguration time due to lower receive power at the receivers, the figure shows two series of violinplots. The

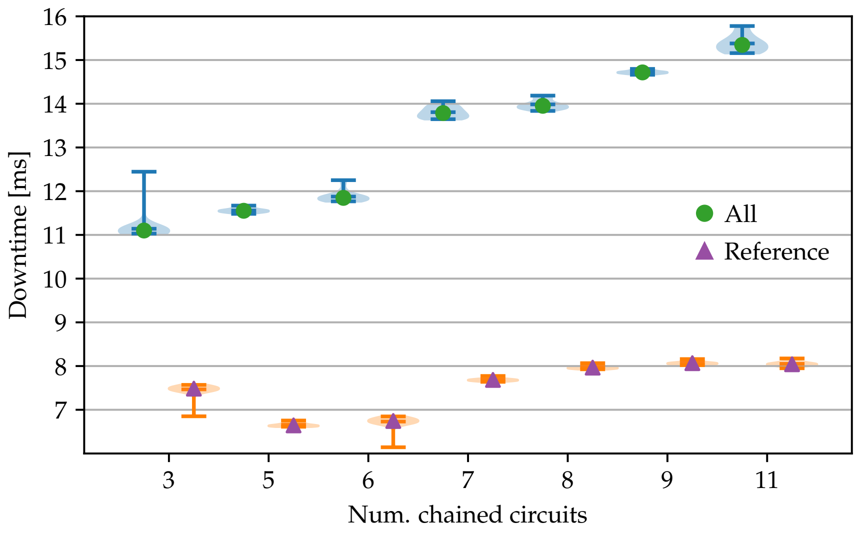


Figure 4.11 Violinplots of the dataplane downtime against the number of chained circuits in the shoelace setup. Downtime is measured with SmartNIC (P4), SI case. Downtime increases with the number of chained circuits.

"*reference*" measurement considers the same length of chained circuits but reconfigures only the last one. The "*all*" series reconfigures all circuits.

The variance is low for all settings, which implies a deterministic behavior given the number of changed circuits. Moreover, the data plane downtime of the reference measurements is relatively stable. Its median values vary between 6.7 ms and 8.8 ms.

However, for changing all circuits in the chain ("all"), an increase of the data plane downtime with the number of chained circuits is evident. Specifically, for chaining only three circuits, the average value is 11.1 ms, while the value is 15.3 ms for chaining 15 circuits — an increase of more than 25% compared to the reference case. While exact reasoning needs a deeper investigation, we conclude that the length of the reconfiguration chain should be considered when modeling end-to-end reconfiguration delay.

Takeaway: Data plane devices react differently to OCS reconfigurations. While the NICs achieve small downtimes, the tested switches with default configurations are less suited for such scenarios. The considered influence factors do not significantly impact the data plane downtime. The data plane component is considered constant. However, we note that chaining multiple reconfigurations increases downtime significantly.

4.5 Evaluating Control Plane Delay

While the downtime of the data plane affects the actual resource usage, it is also important to consider the behavior of the OCS's control plane. Slow reactions here lead to the delayed setting of the new circuits or require issuing the command ahead of time so that changes are in place at the right time. After detailing the setup, this section presents results for a single reconfiguration and then for a sequence of multiple reconfigurations.

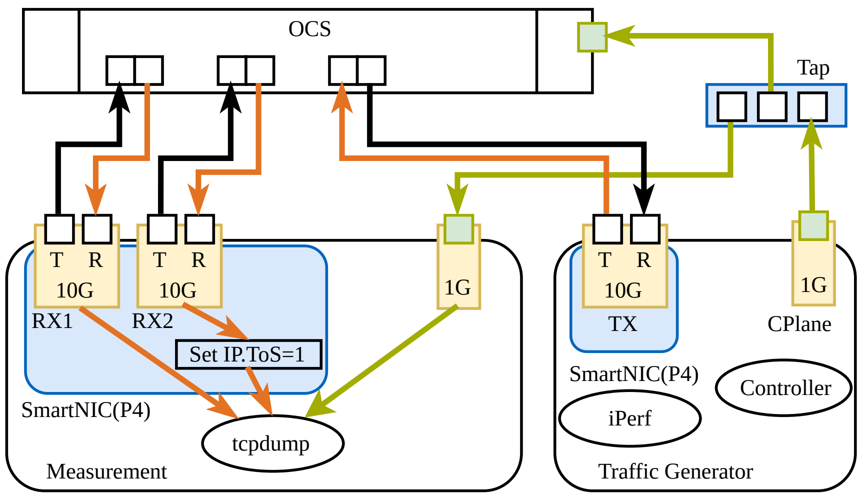


Figure 4.12 Testbed setup for the measurement of the control plane delay. The setup is the same as for the SmartNIC case. In addition, the control plane traffic is tapped and dumped.

4.5.1 Detailed Setup

The measurements of the control plane delay use the same setup as the P4-based measurements with the SmartNIC. In addition to the data plane, also the control plane traffic is captured using a 1G network tap and *tcpdump* (Figure 4.12). As described in Section 4.3.2, the control plane delay can directly be calculated from the timestamps in the packet trace. Comparing measurements of the SmartNIC showed only a small difference ($< 1\%$) between the values obtained from *tcpdump*'s timestamps and those from the P4 program. The evaluated control plane protocols are TL1 [220], a widely used management protocol for the optical domain, and the more generic network management protocols Netconf [221] and Restconf [235].

4.5.2 Single (one-shot) Reconfiguration

First, we consider a single reconfiguration to identify the impact of the used control plane protocol as well as the size of the reconfiguration command, i.e., the number of modified circuits.

4.5.2.1 Control Plane Protocols

Figure 4.13 shows the empirical CDFs of the control plane delays for the three control plane protocols. It also compares the results for running the OCS for > 180 days and for running it only a few days (< 5 days). In both cases, TL1 reacts significantly faster than Netconf and Restconf. After the restart of the OCS (< 5 days), TL1 takes 85 ms on average to apply the changes, while Netconf and Restconf take 245 ms and 257 ms. These values are in the same range as observed in prior studies [45]. Netconf and Restconf obtain almost the same timings. This is because both use the YANG data model of the OCS and apart from the connections, *ssh* for Netconf and *HTTPS* for Restconf, are handled by the same code in the OCS's firmware. All control plane interfaces show low standard deviations of 16.6 ms for TL1, 57.7 ms for Netconf, and 37.6 ms for Restconf.

Comparing the two temporal cases, there is a strong software aging behavior of the OCS: over 180 days of operation, the control plane delay increases by one order of magnitude. While this is

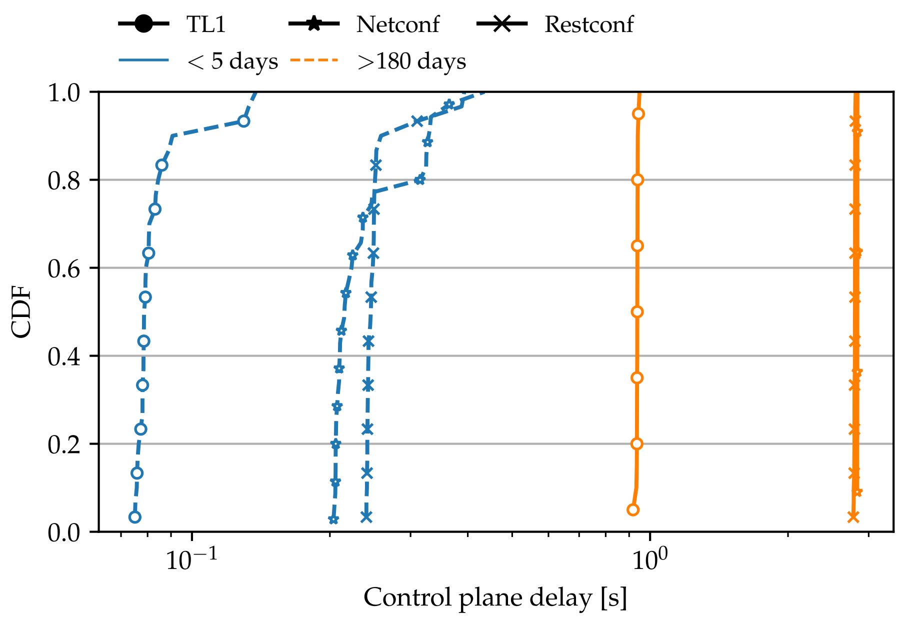


Figure 4.13 cumulative distribution function (CDF) of control plane delay for three control plane protocols. TL1 performs significantly better than Netconf and Restconf.

mainly a quality management issue, operators need to account for or monitor such aspects when deploying *DA* RDCNs [236].

4.5.2.2 Number of Modified Circuits

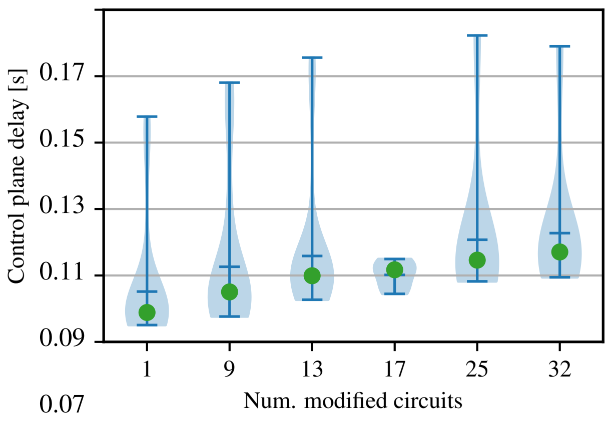
As for the data plane considerations, the number of modified circuits might also impact the control plane delay. Figure 4.14 shows boxplots of the control plane delay against the number of modified circuits. Note that the number of modified circuits correlates to the control plane message size. For TL1, the mean values indicate a steady increase in the delay from around 84 ms for only one modified circuit to around 102 ms when 32 circuits are changed. The behavior seems to be linear, leading to the following hypothesized relationship:

$$C(n) = \delta + \gamma \cdot n. \tag{4.8}$$

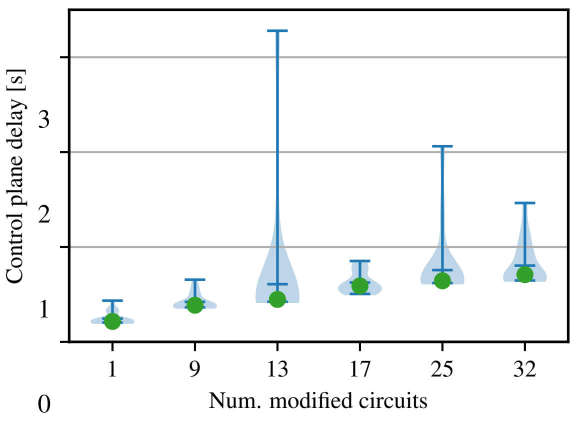
Performing linear regression results in values $\gamma = 0.000561$ and $\delta = 0.086393$ with $p = 0.00341$, which is significant for $\alpha = 0.05$. For Netconf, similar behavior is observable. The obtained parameters from the linear regression are $\gamma = 0.01779$ and $\delta = 0.279702$ with $p < 10^{-10}$. This relationship seems to be intuitive. However, knowing how the control plane behaves opens new possibilities for scheduling reconfigurations and for using available resources more efficiently. Moreover, there are some severe outliers in the data. These indicate unpredictable behavior, which might hinder the application of today's commercially available OCS for frequent reconfigurations. The following part evaluates this in more detail.

4.5.3 Multiple reconfigurations

The obtained models for the control plane delay provide a lower bound on the reconfiguration period possible with the evaluated OCS. However, up to here, the reconfigurations have been



(a) TL1.



(b) Netconf.

Figure 4.14 Boxplots of the control plane delay against the number of modified circuits. For TL1 and Netconf a linear relationship is observable.

performed in a "one-shot" style. Thus, the question arises whether the OCS behaves consistently with periodic reconfigurations, i.e., under constant load.

To assess this, the OCS is stressed with 20 consecutive reconfigurations via the TL1 interface with inter-reconfiguration periods between 0.2 s and 3.0 s. The evaluation focuses on two aspects. First, if reconfigurations are successful, that is, if the number of reconfigurations observed in the trace files matches the number of issued commands to the OCS.

Figure 4.15 illustrates control plane delays (boxplots) and the fraction of successful reconfigurations (line). First, we observe that for low reconfiguration periods, a significant number of reconfigurations is unsuccessful, i.e., not observed in the trace. The average fraction across the runs is below 80% for a reconfiguration period $\leq$ 0.5 s. In addition, it exhibits a high variance. The fraction of successful reconfigurations steadily increases, whereas the variance over the runs reduces for higher reconfiguration periods.

For large reconfiguration periods, the control plane delay is in the expected range from the single-shot measurements, around 100 ms. With decreasing reconfiguration period, the delay and also the number of outliers, i.e., values exceeding 1.5 times the inter-quartile range, increase. Thus, stable operation with low reconfiguration periods is hardly possible with the current firmware of the OCS.

Takeaway: The control plane component of the reconfiguration delay is less deterministic than the data plane one. We observe a significant influence of the command parameterization on the control plane delay. Furthermore, the measurements have larger variances and show a significant number of outliers. Putting modest stress on the control plane results in a loss of reconfigurations. The behavior is explainable with the reliance on software for processing.

4.6 Flexible framework for Experimentation with Reconfigurable Networks (ExReC)

ExReC is a flexible framework for building and evaluating different hybrid reconfigurable topologies. It generally follows the TMT model (cf. Section 3.6 considering a two-layer leaf-spine

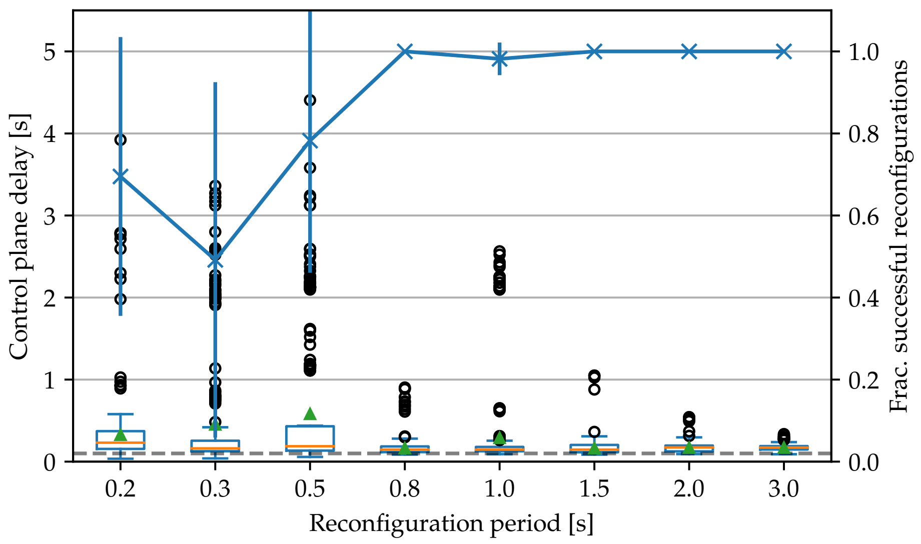


Figure 4.15 Control plane delay against reconfiguration period. For each period, 20 consecutive reconfigurations of one circuit are executed. TL1 is used. The success ratio decreases with the reconfiguration period. At the same time, the control plane delay increases.

structure. The spine layer is constituted by k_{do} *DO* and k_{da} *DA* switches which connect to all leaf nodes.

ExReC is designed to prefer COTS equipment for implementing the *DO* and *DA* switches. However, it can fall back to emulations of the circuit switching behavior if components are not commercially available or do not meet the requirements for frequent reconfigurations [11], e.g., when varying the reconfiguration delay. In its current embodiment, ExReC uses emulation for the *DO* switches. This section first presents an overview of the whole framework and then describes the emulation of *DO* switches and the integration of *DA* switches.

4.6.1 Design Overview

Figure 4.16 shows how ExReC emulates a setup with N racks and ToRs (leafs) and $k = k_{do} + k_{da}$ spine switches. It implements this setup with $M + 1$ physical servers (blue boxes), an EPS, and a COTS OCS (grey boxes). M of the servers emulate the ToRs (red boxes), such that each server emulates $x = \frac{N}{M}$ ToRs and racks.

The servers have enough physical ports to connect to the EPS that emulates k_{do} *DO* switches and to the OCS that implements the *DA* topology part with up to k_{da} switches. An additional control server runs two controller processes: one for the *DO* and one for the *DA* switch. The *DA* process controls the OCS; the *DO* process sends messages to the ToRs to control the *DO* links. The control server connects via a dedicated management network.

ExReC uses QEMU/KVM [237] to spawn virtual machines (VMs) that represent the racks (white boxes). Hosts inside the racks are abstracted: the racks can run either traffic generators like MoonGen [234] and iPerf [232] or applications such as the DML framework Horovod [73] (yellow box “App”). All traffic leaving or entering the VMs is dumped for later analysis. ExReC relies on two design decisions to flexibly evaluate different topology configurations: (1) emulating circuit switching using labels and (2) scheduling traffic directly on the physical servers.

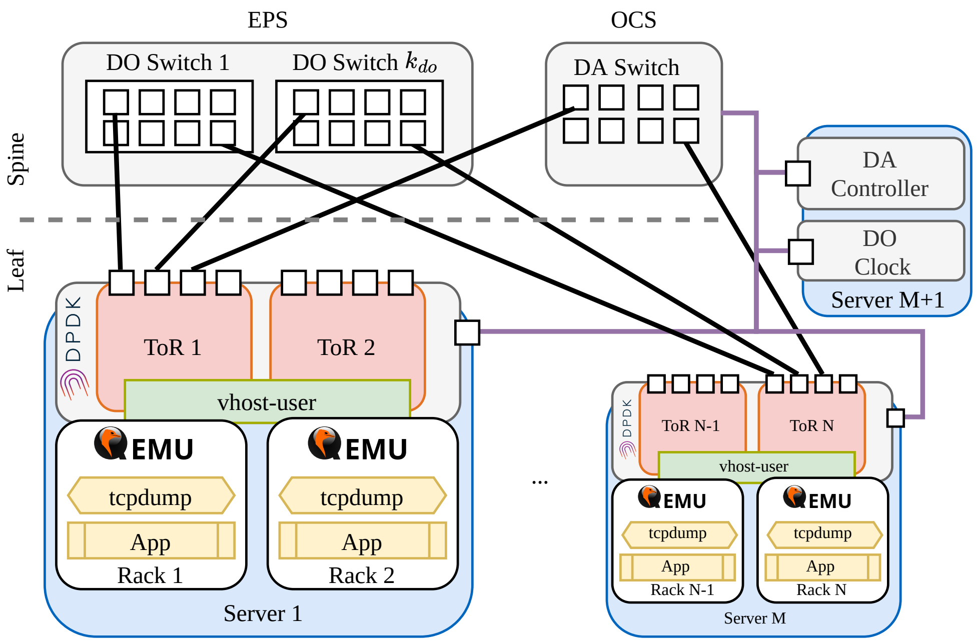


Figure 4.16 Overview of the ExReC framework. An EPS emulates k_{do} *DO* switches whereas a COTS OCS realizes k_{da} *DA* switches. N ToRs are emulated using DPDK on M physical servers. An additional server hosts the controllers.

Label-based routing: COTS equipment for dynamic topologies is hard to obtain. Either it does not meet requirements for reconfiguration times [11], or it is not available, e.g., in the case of *DO* switches [48]. ExReC can emulate a flexible number of *DO* switches among one or many EPSs. Other examples update forwarding entries on OpenFlow-capable EPSs to emulate circuits [51, 191]. However, such switches can behave unpredictably under frequent reconfigurations [215]. Therefore, ExReC emulates the *DO* component with a label-based forwarding approach to achieve correct forwarding with low reconfiguration times. The servers add a label that indicates the destination when sending a packet. The EPS is not reconfigured, which reduces complexity.

Server-based scheduling: *DO* topologies rely on buffering and scheduling logic. Implementing such logic is not possible with today's COTS (programmable) switches, to the best of our knowledge. Therefore, ExReC moves these tasks to the servers: Each server runs a DPDK application emulating the ToRs. The application fetches traffic from the VMs and schedules it on the links to the *DO* and *DA* switches.

4.6.2 Implementation Details

Figure 4.17 illustrates the structure of the DPDK application. Each server runs $3 \cdot x + 1$ threads (rounded rectangles) where x is the number of emulated ToRs per server. For every emulated ToR, there is one `TX1`, one `TX2`, and one `RX` thread. All ToRs on a physical server share one `Sync` thread. The application uses the DPDK *vhost* library to receive and send packets from and to the VMs. The details of the forwarding process for both switch types are described in the following.

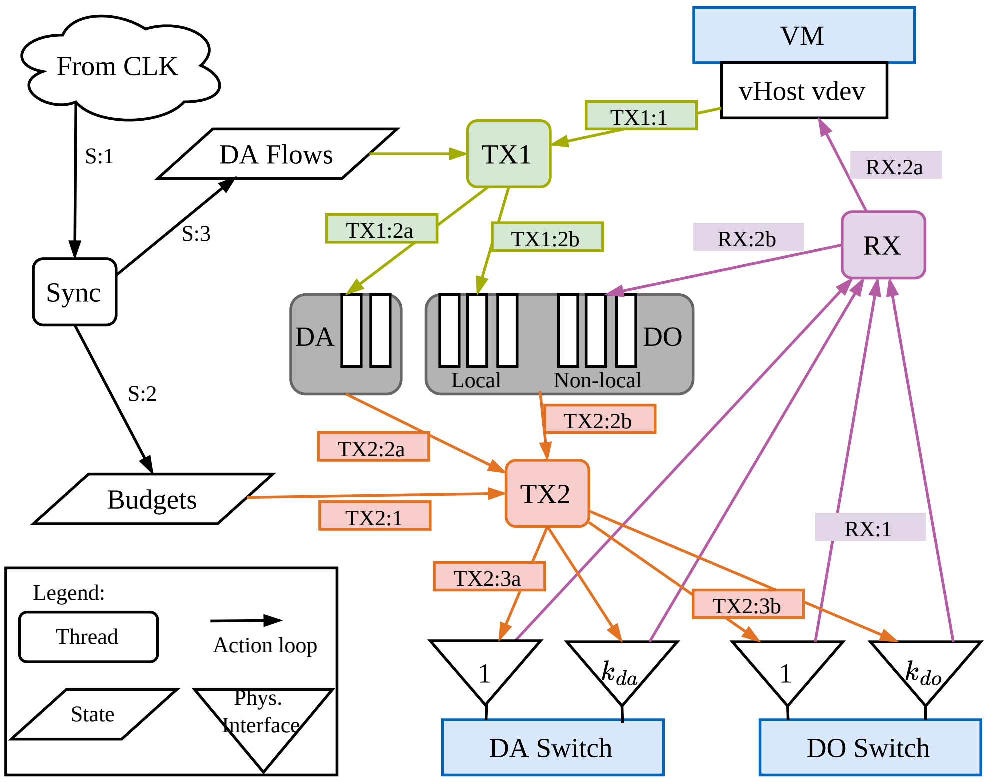


Figure 4.17 Overview of the DPDK-based ToR emulation. Several threads forward packets from and to the VMs and the ports of a ToR. A dedicated thread synchronizes the state of the *DO* switches across the network.

4.6.2.1 Emulating *DO* switches:

The transmission via *DO* switches orients at the design of RotorNet [48]. RotorNet introduces two sets of destination-based queues per ToR to drain packets from after the scheduler ran. Such queueing is not available in COTS switches, another reason for emulation. The DPDK application buffers packets leaving the rack in the "local" queues (`TX1`:2b). For indirect forwarding, packets that are received but are not destined for the ToR (`RX`:2b), i.e., for which the ToR is an intermediate hop, are put to the "non-local" queues.

The label-based routing uses VLAN tags. When sending, the DPDK application first reads per-queue and per-*DO* link budgets (`TX2`:1). The budgets indicate per *DO* link how many packets of each queue can be sent. Then, the thread pulls packets from the corresponding queues (`TX2`:2b) and adds a VLAN tag that indicates the active matching (`TX2`:3b). On the EPS, pre-installed static rules resemble the matchings of the *DO* switches. The EPS matches the VLAN tag and the incoming port and forwards the packet to the corresponding outgoing port. This source routing-based approach eliminates the need for updating rules on the switch and, thereby, reduces the achievable slot size of the *DO* emulation. Upon packet reception (`RX`:1), the DPDK application forwards the packets to the connected racks based on the incoming port of the packet and the active matching (`RX`:2a). If the packet is not destined to the rack, it is put into the respective non-local queue (`RX`:2b).

The `DO Controller` is responsible for cycling through the matchings. It is implemented on top of MoonGen [234]; this provides an easy adaptation of slot sizes and emulated reconfiguration times. The `DO Controller` sends the next active VLAN id to all servers. This way, all ToRs use the same VLAN id for packets to be sent during a slot.

The `Sync` thread reads VLAN ids from the `DO Controller`'s packets (S:1) and updates the active matchings of the ToRs. This, in turn, triggers a recalculation of the budgets (S:2). Modifying the budget calculation can realize different forwarding policies, such as forwarding only directly or performing Valiant load balancing (VLB) [48]. The current state does not synchronize the remaining capacities across the ToRs.

This VLAN-based approach results in little reconfiguration time with respect to the achieved slot size. Therefore, a dedicated VLAN id indicates the reconfiguration of the *DO* switches. When this VLAN id is received, the application sets all budgets to 0, i.e., stops sending until it receives a VLAN id for a valid matching. Each server runs only one instance of DPDK application which handles the traffic for all emulated ToRs with separate queues and threads. This reduces the jitter of the synchronization method.

4.6.2.2 Integrating *DA* switches

While *DO* switches are not yet commercially available, OCSs for realizing *DA* switches are. Several ports on each NIC can connect via fiber to a commercial, full crossbar OCS. Such an OCS can change the connectivity between the ToRs at run-time: it provides bidirectional links that are reconfigured on-demand in approximately $r_{da} \approx 25$ ms [222]. The `DA Controller` activates the optical links at run-time and then updates the *DA* flows inside the DPDK application via control plane messages (S:3). *DA* flows are matched using Layer 3 (source and destination IP addresses) and Layer 4 (source and destination ports) information. Their packets are put into dedicated queues per *DA* switch (`TX1`:2a). Dividing the queues between *DA* and *DO* switches avoids head of line blocking in the queues of the *DO* links. Once packets are enqueued for the *DA* switch (in the corresponding queue), even if there would be an upcoming matching on *DO* switches, packets cannot be sent to a *DO* switch anymore. `TX2` fetches the packets from the queues and forwards them accordingly (`TX2`:2a). All *DA* queues are served in a round-robin fashion, i.e., `TX2` fetches a limited number of packets from each queue. Baseline measurements using MoonGen confirm that this mechanism is enough to saturate the *DA* switch. Moreover, ExReC assures that no other bottlenecks besides the network occur, e.g., by CPU shaping between links in the DPDK application.

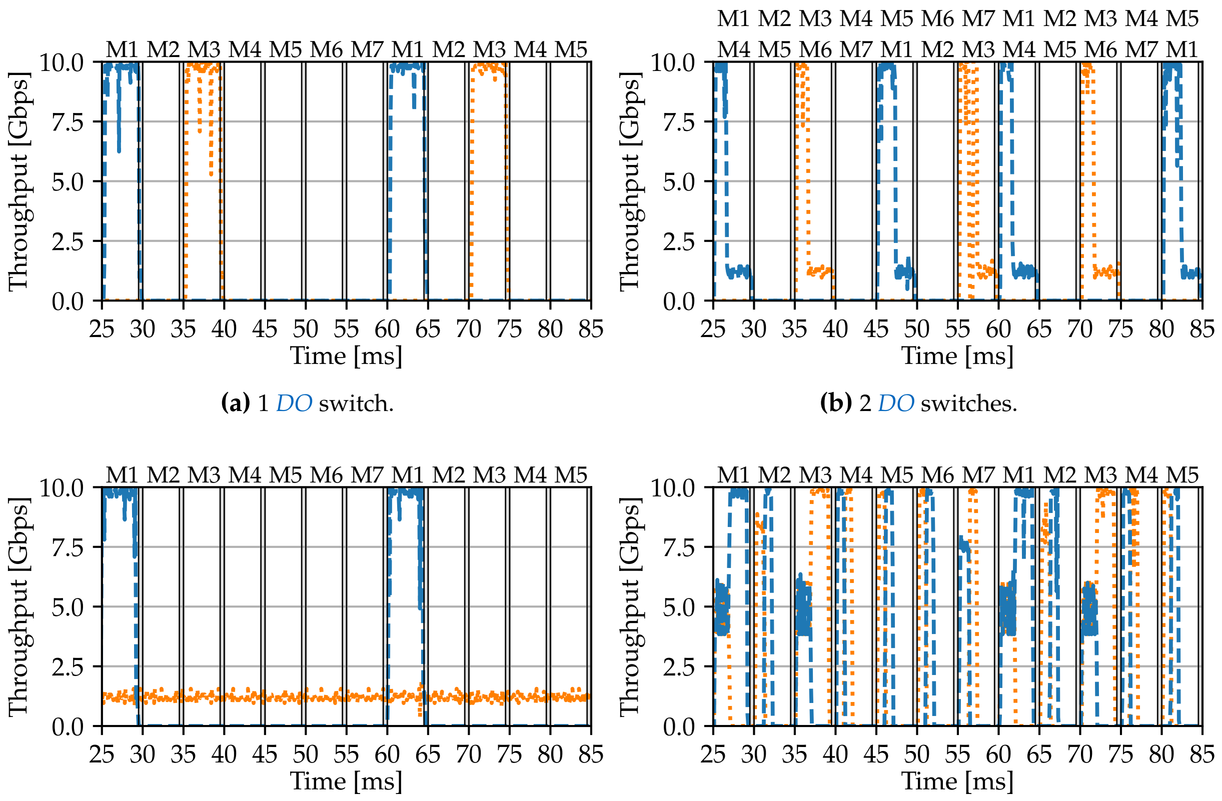
Note that ExReC uses a COTS OCS since it is available. Moreover, using available components generally adds credibility to the measurements. However, in principle, the *DA* switch could also be emulated similarly to the *DO* switches.

4.7 Validation & Evaluation

In order to show the operation of ExReC, this section first gives details on our testbed and the considered settings. Then, it demonstrates (1) validation of our control mechanism, (2) evaluation of different traffics, and (3) an application example using DML training.

4.7.1 Testbed & Settings

The testbed consists of four servers running Ubuntu 18.04 (5.15.0-47-generic kernel) with 128 GB of RAM and Intel Xeon Silver 4114 @ 2.2 GHz (20 cores), which emulate the ToRs. All servers



(a) 1 DO switch.

(b) 2 DO switches.

(c) 1 DO and 1 DA switches.

(d) 1 DO with indirect forwarding.

Figure 4.18 Throughput over time for four configurations. Comparison between 1 DO (a), 2 DO (b), 1 DO and 1 DA (c), and 1 DO with indirect forwarding (d) for two flows to the same destination. A full cycle of the DO switches completes after 7 matchings, i.e., 35 ms.

have two Intel X710-DA4 4x10G NICs [224] and two 1G Intel onboard-NICs (eight 10G ports and two 1G ports). In principle, this allows exploring configurations from fully DO (four DO switches) to fully DA (four DA switches). For DO emulation, the testbed contains a Dell S4048-ON OpenFlow-capable switch [229], and for DA switches, a Polatis Series 6000n 32x32 OCS [222]. The demand-aware `Controller` and the `Clock` run on a dedicated machine. Based on intensive measurements, the most stable achieved inter-arrival time of control messages is ≈ 0.5 ms. Hence, to obtain a duty cycle of 90%, all scenarios use a slot duration of 5 ms composed of an active time of $s = 4.5$ ms and an artificial reconfiguration delay of $r_{do} = 0.5$ ms for DO switches.[6]

4.7.2 Validation of Topology Components

First, a deterministic scenario verifies the behavior of DO and DA switches using four settings: 1 DO switch, 2 DO switches, 1 DO and 1 DA switch, and 1 DO with indirect forwarding. In particular, for DO switches, this demonstrates how the flow rates behave over time when connections between ToRs exist only when matchings are available (cf. Section 3.3.4). For applications that are constantly sending traffic, packets are buffered if the current matchings do not connect the flow's source and destination. The setups consider 8 ToRs so that there are 7 matchings to be cycled through denoted as M1,. . . ,M7.

[6]No cross traffic affects the separated management network; however, slight variances in our clock due to hardware or software interference on the controller machine or switch are possible.

Two MoonGen instances generate two packet streams with a constant rate of 1.25 Gbps. They originate at two different ToRs but have the same destination ToR. Figure 4.18 shows the throughput of the two flows for 60 ms from the steady-state phase. White areas indicate the slots for *DO* topology. The matchings per *DO* switch are shown at the top (M1 - M7). Grey-shaded areas are periods of reconfiguration.

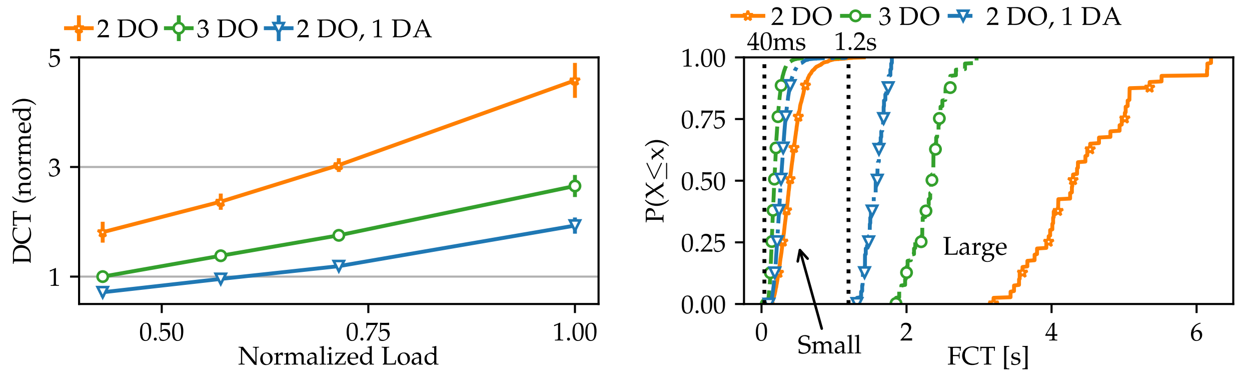
With one *DO* switch (Figure 4.18a), the matching M3 serves the red flow between 35 to 39.5 ms and 70 to 74.5 ms with 10 Gbps. The flow saturates the link for roughly the whole duration of the slot. During the other 6 matchings and the reconfiguration times, packets sent by the source are buffered. The data accumulates to 1.25 Gbps $\cdot$ $(6 + 1) \cdot 5$ ms = 43.75 MBit. This approximately matches the data that can be sent within one slot (10 Gbps $\cdot$ 4.5 ms = 45 MBit). Thus, when M3 is active, the slot is almost fully utilized at 10 Gbps.

After six slots, the red flow sends again with 10 Gbps for the whole slot duration. This shows (1) that *DO* emulation allows applications to send within the available slots and (2) that while there is no connection available, traffic is buffered — the desired behavior. The blue flow has the same behavior as the red flow; however, sending when M1 is active.

With the second *DO* switch (Figure 4.18b), the time until a pair of ToRs has a direct connection again is approximately halved. Again, M3 serves the red flow. During this time, all traffic is sent at 10 Gbps until the buffers are empty. Then, the rate drops to 1.25 Gbps, the rate of the traffic source. The red flow is served from 35 to 39.5 via *DO* 1 (upper row), from 55 to 59.5 ms via *DO* 2, and from 70 to 74.5 ms via *DO* 1 again. From 39.5 to 55 ms, traffic is buffered during three slots (plus the duration of reconfigurations). Between 59.5 and 70 ms, traffic is buffered only during two slots. These different levels of buffer occupancy reflect in the utilization. The rate of the red flow is longer $>$ 1.25 Gbps during the matching from 55 to 59.5 ms. Again, the blue flow shows a similar, shifted behavior.

Figure 4.18c illustrates the effect of having one *DO* and one *DA* switch. The *DO* switch serves the blue flow as expected: traffic is received only when a matching (M1) is established. In contrast, the *DA* switch serves the red flow via a direct link between the source and destination. The flow sends at a constant rate of 1.25 Gbps, which is the configured traffic rate.

Figure 4.18d visualizes 1 *DO* switch with indirect forwarding. Here, the flows are generated with 1.5 Gbps so that the buffered traffic over a cycle exceeds the capacity of a single slot. The budget policy is configured to dedicate 20% of the remaining volume of a slot to send traffic indirectly. As a result, both flows are served in all slots and use only 20% of the slots when sent indirectly. There are two behaviors evident while sending indirect traffic. First, simultaneous sending (M1, M3) and second, sequential sending (M2, M4, M5, M6, M7). In the former case, the flows initially share the rate due to the round-robin-based serving across all queues. After the indirect traffic has been received, the direct traffic uses the full link rate. The effect of sequential matching comes from how packets are stored on the intermediate ToRs. Both flows are received sequentially in the order that packets have been added to the non-local queue of the intermediate ToR. This order varies depending on the active matching. Overall, this demonstrates that the budget calculation function can implement different forwarding policies. A full re-implementation of RotorLB [48] is out-of-scope.



(a) DCT.

(b) FCT by flow size (load = 0.43).

Figure 4.19 Comparison of demand completion time (DCT) (a) and flow completion time (FCT) (b) for three configurations. Vertical lines indicate ideal transmission times via *DO* (small flows) and *DA* switch (large flows).

4.7.3 Measuring & Evaluating Complex Traffic

The subsequent evaluation focuses on more complex traffic scenarios with different traffic intensities and three topology settings (2 *DO*, 3 *DO*, and 2 *DO* & 1 *DA*) with 8 ToRs. For each traffic scenario, it considers two types of flows: small flows and large flows. The small flows have a size of 6.25 MB. Source and destination of the flows are sampled uniformly from the 8 ToRs. These flows should be sent via the *DO* switches since scheduling on a *DA* switch does not amortize due to its larger reconfiguration time. Besides, each experiment has 4 large flows with a size of at least 250 MB. With ideal traffic management, these flows should be transmitted via *DA* switch connections. We vary the arrival rate of all flows and the size of the large ones to increase the load while keeping the ratio between the volume of small and large flows at 2 : 1.

Figure 4.19a shows the DCT (finishing all flows). The figure shows the average value with 95%-confidence intervals against the load. The load is normalized by the maximum stable load. DCTs are relative to the result for 3 *DO* and the smallest load (0.43). For all settings, the DCT increases with increasing load. Adding one *DO* to the 2 *DO* case (3 *DO*) almost halves the DCT for all loads. Moreover, 2 *DO* and 1 *DA* switch, where the *DA* switch specifically serves the large flows, further reduces the DCT by 25% compared to 3 *DO* switches.

Figure 4.19b shows the individual FCTs for a normalized network load of 0.43. The CDFs are separated by flow size, i.e., small flows are the lines on the left, large flows are on the right. As expected, small flows finish all faster than large flows. Using a *DA* switch improves the FCT of large flows. Note the following detail when comparing 3 *DO* to 2 *DO* & 1 *DA*. The topology impacts both flow classes, but the effects are different. For the large flows, 2 *DO* & 1 *DA* performs best providing a constant rate for them. For the small flows, 3 *DO* switches perform best. Here, the additional *DO* switch reduces the waiting time between the slots of a ToR pair from ≈ 17 ms (2 *DO*) to ≈ 12 ms, i.e., by one slot. This may be significant for delay-sensitive flows.

4.7.4 Real Application Traffic: Distributed ML

ExReC can run real applications like the industry-standard DML framework Horovod [73].[7] Each server has only one Nvidia Tesla T4 GPU [238]; hence, this setup considers only 4 ToRs. It

[7] Chapter 2 gives more details on the expected traffic patterns.

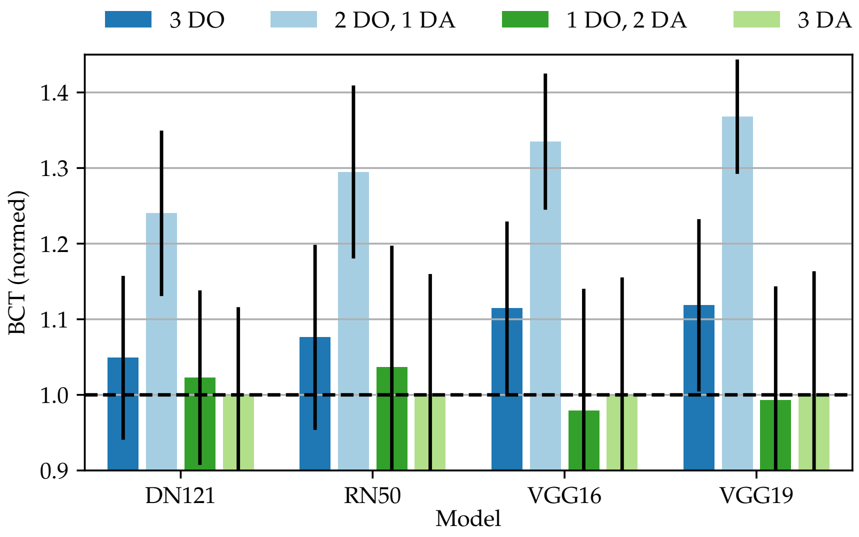


Figure 4.20 Relative impact of configuration on the batch completion time (BCT). Bars indicate mean and 90% confidence intervals. 1 *DO* & 2 *DA* and 3 *DA* have the lowest BCT. These configurations can form a ring for the large flows.

evaluates four machine learning (ML) models of different sizes: DenseNet121 (DN121), ResNet50 (RN50), VGG16, and VGG19 with different topology configurations for 50 batches (training steps) and report the BCT in Figure 4.20. The values are normalized per trained model to the 3 *DA* case, which is a full-meshed network, i.e., optimal here.

The observations across the models are consistent. 3 *DA* obtains the lowest BCT. It is closely followed by 1 *DO*, 2 *DA*, which uses the *DA* links to form a ring that matches the observed traffic pattern of Horovod (cf. Figure 2.4a). All large flows are forwarded via this ring of *DA* links and hence, efficiently served. With 2 *DO*, 1 *DA*, this ring cannot be formed. Here, the BCT is $\approx 22\%$ larger for the smallest model, DenseNet121, compared to the ideal case. Finally, for 3 *DO*, the BCT is lower again. With 3 *DO*, the full-meshed is created again in every slot. However, interruptions due to reconfiguration lead to higher batch durations. The average is increased by $\approx 5 - 10\%$.

4.8 Summary

This chapter provides a deeper understanding of the first characteristic of reconfigurable topologies, the reconfiguration delay. To this end, this chapter studies the modeling, measurement, and emulation of end-to-end reconfiguration delays of programmable optical links using COTS equipment. It presents a measurement methodology and investigates the behavior of five programmable networking devices. The measurements reveal varying performance across the devices with mixed conclusions. In particular, the observed reconfiguration delays from the programmable switches allow only reconfigurations at a low frequency. Furthermore, the analyses show that the specific reconfiguration request impacts the delay on the control plane. Interestingly, many theory-oriented works neglect this variability, i.e., rely on the assumption that end-to-end reconfigurations of optics are deterministic. The measurement results report that reconfiguration times are predictable but not constant — a fact that should be considered in future work.

In addition, this chapter introduces ExReC, a flexible framework for building reconfigurable networks, which only relies on COTS hardware. ExReC can assess the performance of different topologies combining *DO* and *DA* switches. It can flexibly run different combinations of *DO* and *DA* topologies, going from all *DO* to all *DA* with the same testbed setup. Moving the flow scheduling complexity to the physical servers allows controlling the reconfiguration delay. Moreover, it enables the implementation of various forwarding policies for the *DO* part.

A label routing-based emulation replaces *DO* switches that are not commercially available. In principle, this approach can also be applied to the *DA* topology. However, using available components as far as possible reduces the assumptions underlying the measurements. This chapter evaluates ExReC for 8 ToRs only. However, ExReC can evaluate larger settings, e.g., by adding more physical servers or ports on the servers and switches. Higher link rates are constrained by the performance of the CPU.

The evaluation hints at the benefits of different reconfiguration classes for different flows. We will analyze and explore these benefits more in detail in Chapter 7.

Chapter 5

Leveraging Multi-layer Reconfigurability for Flexibility in WANs

Internet Service Provider (ISP) networks form a critical backbone of the digital society. Given the popularity of data-centric applications related to health, science, social networking, business and entertainment, it is expected that the traffic carried by these networks will continue to grow explosively, especially to and from datacenters (DCs) [239, 33]. The COVID-19 pandemic has further emphasized the need for an efficient and reliable communication infrastructure. Over the last years, researchers and service providers have innovated at several layers of the networking stack to improve the efficiency of such infrastructures. The proposals render networks more flexible and *demand-aware* and allow exploiting the specific spatio-temporal structure in the demand. For instance, ad-hoc traffic engineering in wide area networks (WANs) has been replaced with software-defined centralized controllers (e.g., Google B4 [240] and Microsoft SWAN [241]), commodity hardware load-balancers have been replaced with software load-balancers [242, 243], and switch vendors' management APIs have been replaced with in-house switch stacks [41, 231]. With each such technology investment, providers are improving the performance and cost-of-ownership of networks by adding reconfigurability to individual components.

Recently, these innovations also include the (optical) topology layer. As Chapter 3 elaborates, emerging optical technologies allow reconfiguring the network topology in a demand-aware (*DA*) manner within hours, minutes, or even seconds [57, 58, 244, 131]. This, in principle, may further improve the efficiency of networks: by providing "shortcuts" between more frequently communicating sites, the overall traffic may be reduced even in the short term, saving resources and improving latency [60, 245, 58, 62].

However, only little is known today about the potential benefits (e.g., related to performance, energy consumption, capital expenditures (CAPEX), quality of service (QoS)) and limitations of more adaptive optical networks, *jointly* optimizing reconfigurations on multiple layers and hence also accounting for topological flexibility.

This chapter investigates the potential benefits of using reconfigurations on multiple, jointly optimized layers, thereby targeting the second characteristic of reconfigurable topology. In particular, it considers the use case of an eyeball WAN which serves content delivery network (CDN) traffic. The study of CDN traffic is interesting for two reasons: (1) its enormous volume, the traffic of a few so-called *hyper-giants* [13] constitutes the majority of the ISP's workload today, and (2) *hyper-giants* increasingly interconnect with eyeball networks through *multiple* locations, which introduces an optimization opportunity to handle this traffic efficiently.

In this context, this chapter analyzes the benefits of exploiting the reconfiguration flexibilities for an adaptive and demand-aware re-optimization along three fronts: IP topology, routing, and the CDN end-user mapping. Optimization of these layers or of combinations of two layers has already been evaluated in prior works. In contrast, this chapter studies to what extent and how a reconfigurable optical network topology can be combined with a clever request mapping to optimize *hyper-giant* traffic routing in an ISP network.

Therefore, it contributes an optimization framework that, given the point of presences (PoPs) of the *hyper-giants* and the end-users' demands, *jointly* optimizes and adapts the mapping from end-users to the *hyper-giants*' PoPs, the IP layer topology and routing, and the routing in the optical domain. The joint optimization is empirically compared against baselines that optimize only two layers, using empirical data from a large European ISP. The evaluation provides insights into the required amount and frequency of re-optimizations, the predictability of the required reconfigurations, as well as the benefits of adaptive reconfigurations during special events such as the COVID-19 pandemic (2019-2023) [246], under single link failures, and its robustness under less structured demands. Since the analysis is optimistic in that it assumes a cooperative environment, the chapter concludes with a discussion on deployment scenarios and presents avenues for a possible system design.

Content and Outline Section 5.1 describes the scenario and challenges that come from *hyper-giant*-dominated workloads and elaborates on the opportunities in today's increasingly flexible networks. Section 5.2 briefly reviews the related work. Section 5.3 introduces the framework for joint optimization of multi-layer reconfigurability. The performance of the joint optimization in comparison to existing approaches is evaluated using real data from a large European ISP in Section 5.4. Section 5.5 elaborates on trends that support the deployment of the joint optimization approach in CDNs' and ISPs' infrastructures and presents a possible system design. Finally, Section 5.6 summarizes the chapter.

This chapter builds on two prior publications. Specifically, the empirical analysis in Section 5.1.2 re-uses findings presented in a previous conference publication [13, Section 3.2]. The remaining sections in [13] are contributions from the co-authors. The rest of the chapter bases on the journal publication [1]. To foster future work, the framework implementation has been made publicly available at `https://github.com/tum-lkn/hypergiant-isp-optimization`.

5.1 Motivation

This section first introduces more details on the considered scenario. Then, it describes the challenges, opportunities, and enablers for the joint optimization.

5.1.1 Detailed Scenario Description

Figure 5.1 illustrates the considered scenario for a single *hyper-giant*. It shows three layers. The ISP operates a reconfigurable Optical Network (ON) (bottom) to configure an IP topology and routing (middle). The IP topology and routing connect the ISP's customers ("Users") to the *hyper-giant* peering (PNI) at multiple locations (top). The ONs are usually shared using Dense Wavelength Division Multiplexing (DWDM) systems that contain reconfigurable optical add-drop multiplexers (ROADMs). The ROADMs, in turn, provide connectivity between two nodes

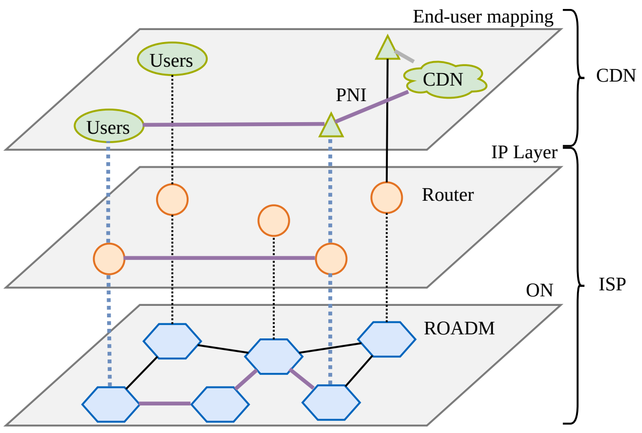


Figure 5.1 Schematic view of the layers of in an ISP-CDN environment. ISPs operate a reconfigurable Optical Network (ON) to connect their customers to content providers such as CDNs. Large CDNs can connect at multiple peering points, which allows the ISP to optimize traffic steering along three dimensions: ON topology, IP layer, and peering point selection.

in the optical topology, by switching/routing lightpaths through the fibers (cf. Section 3.2.2). IP routers connect to the ON via the ROADMs. For every lightpath (solid purple line) that connects two IP routers, a transceiver/port must be available at both ends of the path. A single lightpath provides a fixed capacity. Multiple lightpaths can be aggregated to provide a single IP link with the accumulated capacity of the single lightpaths. By combining multiple IP links, the ISP can build a topology that routes traffic. The traffic enters or leaves the network through the aggregation network towards end-users, or via peerings or internet exchanges towards other networks.

Since large CDNs connect at multiple peering points (green triangles), in principle, operators can optimize the *hyper-giants*' traffic in the ISP's network through clever peering point selection (i.e., mapping end-users to the "best" ingress point) [13]. Assuming a cooperative environment of CDNs and ISPs, this chapter evaluates how CDN peering point selection can jointly be optimized with the ISP network and be adapted towards changing end-user demands over time.

Data source: Tier-1 ISP network. To understand the benefits of jointly leveraging reconfigurability on multiple layers, this chapter provides motivational analyses and evaluations of the algorithms on empirical data from a Tier-1 ISP. The ISP has $>$ 15 million fixed lines and $>$ 30 million mobile users and serves over 50 PB of daily traffic. Its infrastructure features more than 10 PoPs in its home country and overall contains more than 1000 backbone routers. More details on the ISP's profile can be found in [13].

5.1.2 The Challenge: Hyper-giant Traffic

Today, traffic in ISP networks is dominated by a few CDNs or content provider networks. In a previous study [13], we confirm this claim using data from the ISP. We show that traffic of so-called *hyper-giants* makes $\approx$ 75% of the total traffic in the ISP network. Following the definition in our previous work [13], this chapter defines a *hyper-giant* "as any organization that meets the two following conditions: (1) it sends at least 1% of the total traffic delivered to broadband customers in the ISP's network, and (2) publicly identifies itself as a CDN, or is registered as a 'content' or

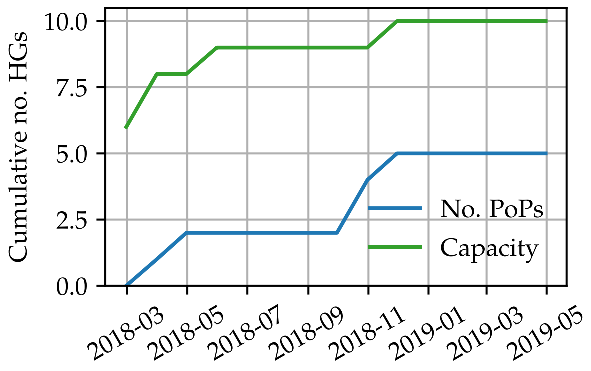


Figure 5.2 Cumulative number of *hyper-giants* with at least one change in their private network interconnects (PNIs) (number of PoPs or capacity). All the top-10 *hyper-giants* increase their capacity within one year and five of them also increased their number of presences.

'enterprise network' in the public database PeeringDB" [13, p. 83]. In the empirical data of the ISP, this definition applies to the traffic of the 10 largest organizations. Moreover, the analysis of traffic from ≈ 2 years of operation shows that the share of *hyper-giant* traffic stays constant meanwhile, the total ingress traffic increases by ≈ 60%. This observation is also confirmed in other studies [239].

Given their size and the critical role *hyper-giants* play today, the efficient delivery of their end-user traffic has become a primary concern of ISPs [247]. An efficient delivery, i.e., providing a good service while keeping network loads low, is in the interest of the *hyper-giants* too, which aim to offer low latency and high quality of experience (QoE) to their customers. Moreover, *hyper-giant* traffic is one of the main reasons for ISPs to upgrade their infrastructures [13].

Today, *hyper-giants* and ISPs often leverage direct connections between their autonomous systems (so-called PNIs), which gives them full control over the links [248]. In fact, to inject traffic into the ISPs network closer to the end-users, ISPs and *hyper-giants* typically connect at *multiple* geographic locations [249, 248]. *Hyper-giants* can control the ingress point to be used for specific end-users, which introduces steerability of the traffic and distinguishes *hyper-giant* traffic from other traffic. The latter is henceforth called background traffic. Although background traffic can be much more volatile and exhibit more dense connectivity, the large volume of *hyper-giant* traffic often dominates in ISP networks.

Reconfiguration of ISPs' topology and routing to accommodate diurnal patterns of background traffic has been shown to be feasible [58]. However, the diversity of ingress points introduces the challenge of mapping end-users to the "best" ingress point and adds a new dimension to the problem [250, 251, 13]. The volume and spatial distribution of end-user demands significantly vary over time, not only due to regular diurnal patterns but also because of large events, e.g., [252, 253, 38] or the COVID-19 pandemic [254, 32]. Moreover, also the PNIs themselves evolve over time.

Figure 5.2 illustrates this evolution. It shows the cumulative number of the top-10 *hyper-giants* with changed connectivity (i.e., adding PNIs) and capacity within a year. In order to identify capacity changes, the analysis uses data from the ISP's *SNMP* system that was aggregated to the median value per month. The value of March 2018 serves as a reference. We observe that all of the top-10 *hyper-giants* increase their peering capacity with the ISP. In addition, five of them also add new presences, i.e., interconnects with the ISP. It becomes evident that ISP and *hyper-giants* continuously invest in the infrastructure, diversifying their interconnection and resulting in more options when optimizing the end-user mapping.[1]

Overall, however, while a fixed user-to-ingress point mapping may work well at one point in time, it can lead to congested peerings or network paths at a different time. On the one hand, due to its sheer volume, a suboptimal topology and routing for *hyper-giants*' traffic results in large overheads for ISPs, e.g., in terms of resource efficiency. On the other hand, the deployed IP topology of the ISP affects the shortest paths between end-users and the *hyper-giants*. Hence, end-user mappings, the ISP's IP topology, and the routing interact, which motivates a joint optimization approach accounting for reconfiguration.

5.1.3 Opportunity: Demand-Aware Re-Optimization

Re-optimizations accounting for the changing demand can be beneficial at different time scales. A first opportunity concerns diurnal traffic patterns in large eyeball networks. Typical traffic patterns show a large difference between the total traffic volume during the peak hour (typically in the evening) and the hour with the smallest amount of traffic (typically at night). An analysis of the traffic volumes arising at the ISP reveals that the difference can be as large as a factor of 7. An interesting use case for more adaptive networks hence regards the joint re-optimization of CDN-user-to-ingress-point mapping and the ISP's IP topology. Re-optimizing the topology frees up capacities on the fibers, enabling the ISP to offer additional, time-of-day-based services. One option is to temporally sell bare lightpaths to customers that want to directly connect their sites during low traffic hours (λ-service), e.g., for synchronization. Additional benefits may arise in terms of energy consumption: transceivers may be switched off, and potentially also IP nodes [129].

A second opportunity for potential savings for the ISP and better performance for end-users stems from joint re-optimization on larger scales, e.g., monthly. To accommodate growing demands, both *hyper-giants* and ISPs continuously invest in server, network, and peering infrastructure (cf. Figure 5.2). Besides a simple upgrade of the capacity at already existing PoPs, this also includes interconnecting at new locations. While *hyper-giants*' mapping systems usually try to leverage these new ingress points, joint re-optimization including topology and routing introduces also cost savings for the ISP, e.g., by using available transceivers and fibers more efficiently.

5.1.4 Enablers: Operational Flexibilities

Four trends that are arising in ISP networks and provide operational flexibility foster the proposed joint reconfiguration:

1. Flexible IP topology. The operation and deployment of lightpaths induce costs for the ISP (e.g., CAPEX for the required transceivers or energy costs for the operation of the lightpaths). Hence, ISPs typically aim to deploy as few lightpaths as possible while serving all demands and accounting for additional requirements, e.g., related to resilience or business policies. Changing the IP topology on a long timescale, e.g., for maintenance or to accommodate organic traffic-growth, is already a common operation (cf. [13, Section 3.3]). Recent developments in optical

[1] A more detailed analysis of the dynamics observed in the ISP's network is given in [13].

switching and advances in software defined networkings (SDNs) in the optical domain render not only long-term but also short-term (re-)deployment of lightpaths feasible [255, 256].

However, the time required for changing IP links is still in the order of minutes as multiple steps are necessary: First, the correct paths, including wavelengths, in the optical domain must be set up. Whereas setting the configuration on a single ROADM is doable within a few seconds [244], setting an entire path (in production settings) can take several minutes [57, 58].[2] Afterwards, interfaces on the IP routers have to be set up, and eventually, updated link characteristics are communicated to the routing entity, e.g., the Path Computation Element (PCE) [257]. Overall, this process can happen on the scale of minutes [58] and results in a first optimization opportunity: A programmable network controller can be used to reconfigure the IP topology throughout the day to optimize it for the changing demands.

2. Flexible traffic engineering. On top of the established IP topology, ISPs typically perform traffic engineering using IP or MPLS routing to avoid congestion in the backbone [258]. Recently, novel source-based routing approaches emerged in the context of Segment Routing (SR). Here, each node and adjacency is assigned a unique label (Segment ID). The edge routers push a stack of Segment IDs to forwarded packets which are then used to successively forward the packet [259]. Deployment of SR policies to the edge routers is often done using PCE [257]. This allows adaptable IP routing without propagating updates through the network and results in reconfiguration delays in the order of seconds [260]. In particular, using Adjacency Segment IDs to describe the path removes the need for re-convergence of the Interior Gateway Protocol (IGP) due to IP topology changes. Reachability information or shortest paths are not maintained on the routers in this case. The central controller configures Segment IDs for added or removed links and can update rules on the edge routers accordingly.

3. Flexible user mapping. Many *hyper-giants* employ clever schemes to map end-user requests to the desired ingress point/server, e.g., [261, 262, 263, 264], often using domain name system (DNS). The *hyper-giant*'s DNS system returns different IP addresses to end-users to route them via a specific peering. Thereby, they can optimize, e.g., the requests' latency or the load of the servers. The benefits of flexible mappings that automatically adapt to the state of the network have been shown to significantly reduce latency for end-users as well as the backbone traffic load for ISPs, especially in CDN-ISP collaboration systems such as PaDIS [250] or FlowDirector [13].

The reconfiguration delays of these mapping systems depend not only on the agreed channel between *hyper-giant* and ISP, e.g., fully automated using Application-Layer Traffic Optimization (ALTO) [265] but also on the time until DNS changes are propagated to the end-users. Recent analyses show that this can be in the order of minutes [266]. This control over the ingress point describes the third enabler for short-term reconfigurability.

4. Centralized control for joint optimization. While the enablers discussed above are promising, the highest benefit can be obtained by *joint* optimization along all three interacting layers, as this chapter later elaborates in the evaluation. For example, changing the user mapping without sufficient capacity on the network path may lead to congestion. The SDN paradigm, which is also

[2]Recently, Hall et al. [131] illustrated avenues to reduce this reconfiguration delay to the order of seconds. However their study is limited to a lab environment.

Table 5.1 Overview of related work on capacity planning and CDN-ISP collaboration.

	IP topology	IP routing/grooming	Demand	Reconfigure over time
[147, 148]	✓	✓	✗	✓
[166, 267, 268]	✓	✗	✓	✗
[62, 63]	✓	✓	✗	✓
[167]	✓	✗	✓	✗
[151, 269]	✓	✓	✗	✗
[270, 271, 272, 273, 274, 250, 13]	✗	✗	✓	✓
This study	✓	✓	✓	✓

emerging in ISP networks [58, 256], is the fourth enabler: with its centralized control, a globalized view of all layers of the network is available, and this problem can be overcome.

5.2 Related Work

This study of how reconfigurable topologies can be used to improve CDN traffic routing in ISP networks builds upon several important existing results in the area of optical networks, CDN-ISP collaboration, and network resource optimization. Table 5.1 summarizes them by indicating the layers considered for joint (re-)optimization. These will be discussed in the following.

Capacity planning and routing in optical networks. Capacity planning is a classic problem in optical networks, and there already exists a large body of literature also accounting for multi-layer aspects (cf. Section 3.4). Much existing related work on optical network optimization revolves around the impact of optical network reconfiguration on the routing stability [147, 148], the reconfiguration and adaption of virtual topologies that overlay optical networks to changing traffic demands [61], issues related to incremental deployment [245, 275], as well as regenerator placement problems (i.e., computing ROADM locations) [276]. All these works consider the demand given by source-destination pairs and do not shape it via optimizing end-user mappings like the approach presented in this chapter does.

Another path of research focuses on content-oriented and application-aware optical network optimization [166, 267] and multi-layer resource allocation [62, 63], e.g., in the context of the Facebook network [268]. A case study related to CDNs is conducted in [167], however, without considering reconfigurations. In particular, these works consider optimizations on the demand level but optimize only 2-layers neglecting the flexibility of IP grooming [267, 167], and they do not adapt over time [166, 267, 268]. Adaptive operation is provided by [62, 63] along with the integration of application requirements such as latency constraints. However, demand is provided again by fixed source-destination pairs and is not steerable. While several efforts exist to render operations of optical wide-area networks more adaptive [151, 269], there is no related work on optimizing ISP networks along all three dimensions arising in the context of CDNs: topology, routing, and end-user mapping. In particular, the few existing 2-layer solutions which account for multiple mapping locations, such as [267], do not support IP grooming.

Existing proposals from the DC scenario are not applicable here since the constraints in DCs are fairly different from the ones in ISPs (e.g., in terms of routing [141], availability of wireless channels [47], and concerning workloads [34]).

CDN-ISP collaboration. There is interesting work on how collaborations between CDNs and ISPs can be improved. Such work, however, mainly focuses on traffic engineering (TE), neglecting the potential of topology modification. For related work on joint content placement and TE in this context, see [277, 270, 271, 272, 273, 274]. All this work is limited to user mapping and TE but ignores adaptive optimization of the ISP's topology. A recent case study considers the joint content distribution and TE of adaptive videos in telco-CDNs [278], assuming a caching system where CDNs can deploy media objects.

In contrast, this chapter considers a scenario where requests are handed over at peerings to the CDN. The works closest to this scenario in the context of CDN-ISP collaboration are PaDIS [250] and FlowDirector [13]. Both systems collect topology and routing information from the ISP network to create a ranked mapping between the IPs of the ISP's users and the CDN's servers, e.g., based on geographical distance or the number of hops in the network. The ranked mapping is an additional input to the CDN's mapping system to improve content delivery by serving via servers at closer locations and shorter paths. PaDIS and FlowDirector, aid the CDNs' mapping systems but only collect information from ISP's network and do not adapt it.

Optimization and resource allocation. The optimization problem considered in this thesis is related to the virtual network embedding problem, which has been studied in the context of optical networks as well [159, 160, 161, 158, 162, 163, 164]. See [157] for an overview of existing solutions in this context. In virtual network embedding, both virtual nodes and virtual links are to be allocated. However, in the present case, node allocations are already *given* (namely, the end-user locations and the hyper-giant's peering locations). Only the topology is *flexible*, i.e., to be allocated. This link-only embedding problem boils down to routing, but does not support selecting multiple (content) locations. That said, the approaches in [159, 161, 162] for supporting the addition and removal of IP links could be reused in the CDN-ISP setting; however, these approaches do not consider re-embedding of virtual links, which maps to re-routing of demands. Finally, also works on mapping service function chains on optical networks, e.g., [279], are different from the CDN-ISP scenario as the optimization regards the mapping, not the topology.

5.3 Joint Optimization Framework

In order to study the potential benefits of re-optimizing reconfigurable networks on multiple layers, this section formulates a joint optimization framework. Given the *hyper-giant* and background traffic, the goal is to find assignments of end-users to *hyper-giant* peering locations, the routing on the IP layer, the capacities of the IP links, and their routing in the optical domain. The objective is to minimize the network capacity and to account for re-configurations. This is subject to fulfilling all demands without violating capacities of peering, IP or optical (wavelengths on fibers) links. The presented approach combines optimizations on single layers and *jointly* solves the following sub-problems: (1) assignment of end-user nodes to *hyper-giants*' ingress PoPs, (2) design of the IP topology (connectivity & capacity), (3) selection of optical paths for the IP links, and (4) routing of *hyper-giants*' and background demands in the IP topology. This modeling can leverage the flexibility of the three layers, e.g., demand can be routed via a dedicated IP link and optical path, or can use multi-hop IP routing to reduce resource fragmentation. In general, the optimization provides a solution for a single time instance t.

This section starts by describing the general joint optimization framework (Section 5.3.1). It then presents a greedy algorithm for the end-user mapping (Section 5.3.2). Finally, it introduces the specific optimization algorithms which come in different flavors, highlighting various aspects relevant in our evaluation (Section 5.3.3).

5.3.1 Mathematical Model

Table 5.2 summarizes the sets and functions, Table 5.3 the constants, and Table 5.4 the decision variables. In the following, the time index t is ommitted for sake of readability if it is not needed.

Sets and mappings. The topology consists of two sets of nodes, IP nodes (or routers) ($\mathcal{N}^{\mathrm{IP}}$) and optical nodes ($\mathcal{N}^{\mathrm{Opt}}$). They are collocated and connected via optical transceivers. The mapping from IP node to an optical node is fixed and given by $o : \mathcal{N}^{\mathrm{IP}} \rightarrow \mathcal{N}^{\mathrm{Opt}}, e \rightarrow o(e)$. Multiple IP nodes can be collocated with one optical node. For instance, there can be one IP node for peering with *hyper-giants* and another one for end-user connectivity. The demand layer is described by the set of *hyper-giants* ($\mathcal{H}$) which, in turn, are specified with a set of end-user nodes ($\mathcal{U}_h$) and peering nodes ($\mathcal{P}_h \subset \mathcal{N}^{\mathrm{IP}}$). The mapping from end-user nodes to IP nodes is fixed and given by $i : \bigcup_{h \in \mathcal{H}} \mathcal{U}_h \rightarrow \mathcal{N}^{\mathrm{IP}}, u \rightarrow i(u)$. An IP node can provide connectivity for end-users or peering nodes but not for both, i.e., $\{e \in \mathcal{N}^{\mathrm{IP}} : i(u) = e \; \exists u \mathcal{U}_h\} \cap \mathcal{P}_h = \emptyset \quad \forall h \in \mathcal{H}$. The set of pre-calculated equal-cost candidate paths between IP nodes e and f in the optical domain is denoted by $P_{e,f}$. $P_{e,f}$ already implicitly considers the mapping from IP to optical nodes.

Constants. The constant d_{uh} denotes the demand of end-user node u toward *hyper-giant* h. Different to others, e.g., [267], this model does not differentiate between up- and down-link demands (request and reply). The *hyper-giants* are modeled as super-sources (abstract nodes). An actual peering between the ISP and a *hyper-giant* is modeled with a link between a peering node and the super-source. The peering capacity between the ISP and *hyper-giant* h at peering node $p \in \mathcal{P}_h$ is c_p^h. The constant d_{ab}^{bg} indicates the amount of background traffic between $a, b \in \mathcal{N}^{\mathrm{IP}}$. All three values are given as rates (in Gbps). Each IP router supports a maximum number of transceivers t_e^{max} which each provides a capacity of C. The optical system restricts the number of lightpaths (wavelengths) on a fiber between m and $n \in \mathcal{N}^{\mathrm{Opt}}$ to w_{mn}. The constant l_p denotes the length of a lightpath $p \in P_{e,f}$ in kilometers. In order to allocate some headroom of capacity on the IP links for sudden (small) demand variations, u_{max} defines the maximum link utilization. ρ is the fraction of IP links that may be reconfigured between two optimizations with respect to the $|\mathcal{N}^{\mathrm{IP}}|^2$.

Variables. The binary variable $\hat{o}_{u,ph}$ indicates if demand u from *hyper-giant* h traverses peering node $p \in \mathcal{P}_h$, i.e., it represents the end-user mapping. Another set of variables $o_{u,ef}^h$ indicates the routing of a *hyper-giant*'s demand in the IP layer ($e, f \in \mathcal{N}^{\mathrm{IP}}$). If the demand u from *hyper-giant* h flows over an IP link between e and f the value is 1. Similarly, $o_{ab,ef}^{bg}$ indicates if background demand from a to b traverses IP link from e to f. The optimization considers single path routing. Hence, all these groups of variables are binary valued.

The number of lightpaths that connect IP routers e and f while using path $p \in P_{e,f}$ is given by $y_{ef,p}$. By integrating the path p, $y_{ef,p}$ already determines the routing in the optical domain.

Table 5.2 Sets and Functions.

Notation	Description
$\mathcal{N}^{\mathrm{IP}}$	IP nodes (routers): At every PoP, there is at least one IP router.
$\mathcal{N}^{\mathrm{Opt}}$	Optical nodes (ROADM): At every PoP, there is one optical node which is connected to the IP router at that PoP.
$o(e)$	$\mathcal{N}^{\mathrm{Opt}} \rightarrow \mathcal{N}^{\mathrm{IP}}$, mapping between IP and optical nodes.
$\mathcal{H}$	Hyper-giants: Entities that are responsible for the majority of the traffic.
$\mathcal{U}_h$	End-user demands of *hyper-giant* h.
$\mathcal{P}_h \subseteq \mathcal{N}^{\mathrm{IP}}$	*Hyper-giants'* peering locations: IP nodes where the ISP connects via PNIs to the *hyper-giants*.
$i(u)$	$\bigcup_{h\in\mathcal{H}} \mathcal{U}_h \rightarrow \mathcal{N}^{\mathrm{IP}}$, mapping between end-user demand and IP nodes.
$P_{e,f}$	(Pre-calculated) Equal-cost paths between $e, f \in \mathcal{N}^{\mathrm{IP}}$ over the ON.

Table 5.3 Constants.

Notation	Description
d_{uh}	*Hyper-giant* demand volume: Aggregated demand from end-user node u (entering at IP node $i(u)$) towards $h \in \mathcal{H}$ in Gbps.
c_p^h	Peering Capacity: Bandwidth of the PNI between ISP and h at $p \in \mathcal{P}_h$ in Gbps.
d_{ab}^{bg}	Background demand: Demand between IP routers $a, b \in \mathcal{N}^{\mathrm{IP}}$ in Gbps.
t_e^{max}	Number of transceivers at IP router $e \in \mathcal{N}^{\mathrm{IP}}$.
C	Capacity of a lightpath in Gbps
w_{mn}	Fiber capacity: number of lightpaths that can be allocated on fiber between $m, n \in \mathcal{N}^{\mathrm{Opt}}$.
l_p	Length of path $p \in P_{e,f}$ in km.
u_{max}	$\in [0, 1]$ Maximum allowed IP link utilization.
ρ	Fraction of allowed reconfigurations.

Thus, specific variables for optical routing are not required. Variables r_{ef} indicate if capacity of link $e, f \in \mathcal{N}^{\mathrm{IP}}$ changed compared to the previous timestamp.

Constraints. The optimization is subject to several constraints which address flow conservation, demand fulfillment and capacity limitations on three layers. Besides, limitations in the number of transceivers per IP node as well as bi-directionality of IP links and maximum link utilization are considered:

$$\sum_{p\in\mathcal{P}_h} \hat{o}_{u,ph} = 1 \qquad \forall h \in \mathcal{H},\ u \in \mathcal{U}_h \tag{5.1}$$

$$\sum_{f\in\mathcal{N}^{\mathrm{IP}}:\ f\neq p} \left(o_{u,pf}^{h}\right) - \hat{o}_{u,ph} = 0 \qquad \forall p \in \mathcal{P}_h,\ u \in \mathcal{U}_h,\ h \in \mathcal{H} \tag{5.2}$$

$$\sum_{f\in\mathcal{N}^{\mathrm{IP}}:\ f\neq e, f\notin\mathcal{P}_h} o_{u,ef}^{h} - o_{u,fe}^{h} = 0 \qquad \forall e \in \mathcal{N}^{\mathrm{IP}} : e \neq u \wedge e \notin \mathcal{P}_h,\ u \in \mathcal{U}_h,\ h \in \mathcal{H} \tag{5.3}$$

Table 5.4 Variables

Notation	Description
$\hat{o}_{u,ph}$	= 1 if demand u to h flows over peering node $p \in \mathcal{P}_h$ to *hyper-giant* h.
$o^h_{u,ef}$	= 1 if demand from u to h flows over IP link e, f. $\forall e, f \in \mathcal{N}^{\text{IP}}$.
$o^{bg}_{ab,ef}$	= 1 if demand from a to b flows over IP link e, f. $\forall e, f \in \mathcal{N}^{\text{IP}}$.
$y_{ef,p}$	IP trunk capacity: $y_{ef,p} \in \mathbb{N}$ indicates the number of lightpaths between $e, f \in \mathcal{N}^{\text{IP}}$ using path $p \in P_{e,f}$ and thereby the capacity ($\cdot$ C) of the trunk.
r_{ef}	=1 if capacity of IP link changed in comparison to value from previous instance (y^{t-1}_{ef}).

$$\sum_{f \in \mathcal{N}^{\text{IP}}:\, f \neq u} o^h_{u,i(u)f} - o^h_{u,fi(u)} = -1 \qquad \forall u \in \mathcal{U}_h,\ h \in \mathcal{H} \tag{5.4}$$

$$\sum_{u \in \mathcal{U}_h} \hat{o}_{u,ph} \cdot d_{uh} \leq c^h_p \qquad \forall p \in \mathcal{P}_h,\ \forall h \in \mathcal{H} \tag{5.5}$$

Equations 5.1 through 5.4 describe the flow conservation for *hyper-giant* demands and thereby address sub-problems (1) and (4). Specifically, Equation 5.1 sets the fractions of routed demand per *hyper-giant*-end-user pair to be equal to 1, i.e., all demands from *hyper-giants* (super-source) to end-users must be served. At all IP nodes where neither the end-users nor the peering are connected, ingressing and egressing flow of one demand must be equal (Equation 5.2 and Equation 5.3). At the destination node of a demand, it must sink (Equation 5.4). Equation 5.5 limits capacity of peerings, i.e., the edges from the peering nodes to the super-source node of the respective *hyper-giant*.

$$\sum_{f \in \mathcal{N}^{\text{IP}}:\, f \neq a} o^{bg}_{ab,af} = 1 \qquad \forall a, b \in \mathcal{N}^{\text{IP}} \tag{5.6}$$

$$\sum_{f \in \mathcal{N}^{\text{IP}}:\, f \neq e} o^{bg}_{ab,ef} - o^{bg}_{ab,fe} = 0 \qquad \forall a, b, e \in \mathcal{N}^{\text{IP}}, e \neq a, e \neq b \tag{5.7}$$

$$\sum_{e \in \mathcal{N}^{\text{IP}}:\, e \neq b} o^{bg}_{ab,eb} = -1 \qquad \forall a, b \in \mathcal{N}^{\text{IP}} \tag{5.8}$$

Equations 5.6 through 5.8 are similar flow conservation constraints for background demands. They ensure that flows originate at the sources, terminate at the sinks and that ingressing and egressing flows at intermediate nodes are the same.

$$\begin{aligned} \sum_{h \in \mathcal{H}} \sum_{u \in \mathcal{U}_h} o^h_{u,ef} \cdot d_{uh} + \sum_{a,b \in \mathcal{N}^{\text{IP}}} o^{bg}_{ab,ef} \cdot d^{bg}_{ab} \\ \leq C \cdot u_{max} \cdot \sum_{p \in P_{(e,f)}} y_{ef,p} \\ \forall e, f \in \mathcal{N}^{\text{IP}} : e \neq f \end{aligned} \tag{5.9}$$

$$y_{ef,p} = y_{fe,p} \qquad \forall e, f \in \mathcal{N}^{\text{IP}},\ p \in P_{e,f} \tag{5.10}$$

$$\sum_{f \in \mathcal{N}^{\mathrm{IP}}} \sum_{p \in P_{(e,f)}} y_{ef,p} \leq 2 \cdot t_e \qquad \forall e \in \mathcal{N}^{\mathrm{IP}} \tag{5.11}$$

Equation 5.9 ensures that the demand flowing via IP link e, f does not exceed the given maximum IP link utilization and thereby, addresses sub-problem (2). It considers only the edges adjacent to e. Equation 5.10 and 5.11 account for bidirectionality of IP links, a lightpath is required for each direction, and limit the number of available transceivers per IP node (degree bound) respectively.

$$\sum_{e,f \in \mathcal{N}^{\mathrm{IP}}} \sum_{p \in P_{e,f}:(m,n) \in p} y_{ef,p} \leq w_{mn} \qquad \forall m, n \in \mathcal{N}^{\mathrm{Opt}} \tag{5.12}$$

Equation 5.12 limits the number of lightpaths that can be routed over a fiber and thereby addresses sub-problem (3). Note that the set of relevant paths, i.e., paths which use the optical edge m, n, can be pre-calculated.

$$\sum_{f \in \mathcal{N}^{\mathrm{IP}}} \left\lceil \frac{\sum_{h \in \mathcal{H}} \sum_{u \in \mathcal{U}_h : i(u) = f} d_{uh}}{C \cdot u_{max}} \right\rceil \leq \sum_{e,f \in \mathcal{N}^{\mathrm{IP}}} \sum_{p \in P_{e,f}} y_{ef,p} \tag{5.13}$$

Equation 5.13 provides a lower bound for the objective to reduce the run-time of the solver. In the ideal case, i.e., with minimum resource fragmentation, an IP node with end-user demands is directly connected via a single link to a peering node which is able to serve all these demands. The necessary capacity of such a link is given by summing the demands of all *hyper-giants* at that IP node and accounting for the maximum IP link utilization. Since the optimization is subject to resource fragmentation and requires connectivity to several peerings of the *hyper-giants*, the sum of the capacities of all these ideal links provides a lower bound for the total capacity.

$$\sum_{p \in P_{(e,f)}} y_{ef,p} - y_{ef}^{t-1} \leq r_{ef} \cdot Q \quad \forall e, f \in \mathcal{N}^{\mathrm{IP}} \tag{5.14}$$

$$y_{ef}^{t-1} - \sum_{p \in P_{(e,f)}} y_{ef,p} \leq r_{ef} \cdot Q \quad \forall e, f \in \mathcal{N}^{\mathrm{IP}} \tag{5.15}$$

$$\frac{1}{|\mathcal{N}^{\mathrm{IP}}|^2} \cdot \sum_{e,f \in \mathcal{N}^{\mathrm{IP}}} r_{ef} \leq \rho \tag{5.16}$$

Equations 5.14-5.16 limit the number of reconfigurations in terms of increased or decreased capacity per IP link. Equation 5.14 pushes r_{ef} up if the capacity increases in comparison to the capacity of the previous timestamp y_{ef}^{t-1}. Similarly, Equation 5.15 addresses capacity decreases. Equation 5.16 limits the total reconfigurations over all IP links.

Objective. The main objective is to minimize the total deployed capacity which is given by the sum of capacities of all IP links:

$$\min \sum_{e,f \in \mathcal{N}^{\mathrm{IP}}} \sum_{p \in P_{(e,f)}} y_{ef,p}. \tag{5.17}$$

This objective function serves as a proxy for operational expenditures (OPEX) such as power consumption and CAPEX (e.g., transceivers to buy in long term). Additionally, it hints at the utilization of the optical topology that might in turn be used to operate λ-service during low traffic hours.

5.3.2 Greedy End-user Mapping

The Mixed Integer Program (MIP) presented in the previous section solves the sub-problems (1) – (4) simultaneously. In order to quantify the benefits of this joint optimization, we separate problem (1) and solve it with a greedy algorithm. The resulting end-user mapping is used as input for the MIP which solves sub-problems (2) – (4). This study limits the evaluation to the separation of sub-problem (1) since it splits along the boundary between CDN and ISP. Moreover, this split first makes the demand more specific by fixing the source-destination-pairs for the *hyper-giant* demands and then optimizes the ISP's network. Other splits, e.g., optimizing the IP topology first, would counteract the demand-aware nature of the approach.

Algorithm 1 shows a greedy assignment procedure for a single *hyper-giant* based on the shortest path length between end-users and peering PoPs in the optical topology. The algorithm starts by sorting the end-users PoPs of this *hyper-giant* h by their demand volumes in non-increasing order (l. 1). For every end-user node u, it iterates over the peering nodes of this *hyper-giant* sorted by their distance to the end-user node in question (l. 4f). If the peering node p has enough remaining peering capacity and the number of transceivers at suffices to accommodate the total assigned demand, u is assigned to p, i.e., $\hat{o}_{u,ph} = 1$ (l. 6). Otherwise, the next peering node is evaluated. The algorithm fails if not all demands can be assigned. Algorithm 1 is repeated for all *hyper-giants*.

Algorithm 1 Greedy end-user mapping for a single *hyper-giant* h.

```
Input: $c_p^h, d_{uh} \quad \forall u \in \mathcal{U}_h, p \in \mathcal{P}_h$
Output: $\hat{o}_{u,ph}$
 1: sort $u \in \mathcal{U}_h$ by their demand volume $d_{uh}$ in non-decreasing order
 2: $allocDemand[p] \leftarrow 0 \quad \forall p \in \mathcal{P}_h$
 3: for $u \in \mathcal{U}_h$ do
 4:     sort $p \in \mathcal{P}_h$ by distance to $u$ in non-decreasing order
 5:     for $p \in \mathcal{P}_h$ do
 6:         if $allocDemand[p] + d_{uh} \le c_p^h \land \left\lceil \frac{allocatedDemand[p]+d_{uh}}{C} \right\rceil \le \frac{1}{2} t_p$ then
 7:             $\hat{o}_{u,ph} \leftarrow 1$
 8:             $allocDemand[p] \leftarrow allocDemand[p] + d_{uh}$
 9:         else
10:             $\hat{o}_{u,ph} \leftarrow 0$
```

5.3.3 Optimization Flavors

Considering reconfigurability on different layers, i.e., for the sub-problems (1)–(4), results in four different optimization flavors which are later compared in the evaluation. Note that each flavor requires some adaptation of the MIP, i.e., to fix certain decision variables:

Baseline uses end-user mapping, IP layer topology and routing from the ISP's real data, without performing any additional optimization; it determines the capacity of the IP links to serve all demands. Note that the topology and routing might include some requirements for particular links due to business policies of the ISP.

ISP-only uses the end-user mapping extracted from the real data and optimizes the IP topology and routing with fixed end-user mapping. This ISP-only approach reflects a scenario where either no ISP topology-aware end-user mapping system is in place or the CDNs are not cooperating.

2-step optimizes the end-user mapping using Algorithm 1. In a second step, it optimizes IP topology and the routing using the MIP given the greedy end-user mapping. Thus, CDNs are cooperative but optimization is done separately in two steps.
Joint jointly optimizes all three layers, end-user mapping, IP topology and routing, using the exact approach from Section 5.3.1.

5.4 Evaluation

This section empirically answers the fundamental question: can we leverage reconfigurable topologies to improve ISP traffic steering and resource management? To do so, it first outlines the attributes of the optical infrastructure, the traffic, and the optimization approaches (Section 5.4.1). It then assesses to what extent we benefit from reconfigurations – if at all – and what frequencies of reconfigurations are needed (Section 5.4.2). Further, it complements this analysis by evaluating the benefit of jointly optimizing along all available dimensions (Section 5.4.3) and dissects the costs in terms of observed reconfigurations concerning the topology's recurrence and stability (Section 5.4.4). The evaluation concludes by considering how the results generalize to specific scenarios. To this end, it investigates the approach in the context of the traffic during the COVID-19 pandemic, a scenario with randomized demands, and a scenario with link failures (Section 5.4.5-Section 5.4.7).

5.4.1 Settings

This section details the settings for the empirical evaluation. It first describes the physical infrastructure, followed by the demands and the assessed optimization approaches.

ISP infrastructure. The network considered for the evaluation bases on the structure and data from the production network of a Tier-1 ISP (cf. Section 5.1.1). The ON of the ISP consists of < 20 nodes and < 30 edges.[3] Each edge has an optical capacity with $w_{mn} = W = 100$ lightpaths (wavelengths). The access routers of the end-users have a fixed connection to the core routers, i.e., the ON can only be reconfigured in the core part of the network. Hence, the access routers are aggregated into the corresponding core routers. This reduces the original set of IP routers (> 1 k) to the core routers and the largest peering routers resulting in < 30 IP routers in total. While this reduction roots in the capabilities of the network, it also makes the optimization more tractable. At some PoPs, multiple IP nodes are connected to the optical node. Every IP node can support $t_e^{max} = 100$ transceivers. Each transceiver can create a single lightpath with a capacity of $C = 100$ Gbps. Moreover, the maximum IP link utilization is limited to $u_{max} = 50\%$. This shall provide slack capacity in the network to accommodate traffic in cases of failures. The *hyper-giants'* peering locations and capacities are also extracted from the ISP's data.

Traffic (Demands). We create the demands using real traffic data from the FlowDirector system [13]. FlowDirector gathers > 45 billion NetFlow records per day at the border routers of the ISP. Using the IP addresses and the associated autonomous systems, the traffic data is differentiated into two types of demand samples: `HT` and `BT`. `HT` contains only flows belonging to the top-10 *hyper-giants* in the ISP's network. It makes $> 75\%$ of the total traffic in the ISPs's

[3]Exact numbers cannot be provided due to data privacy.

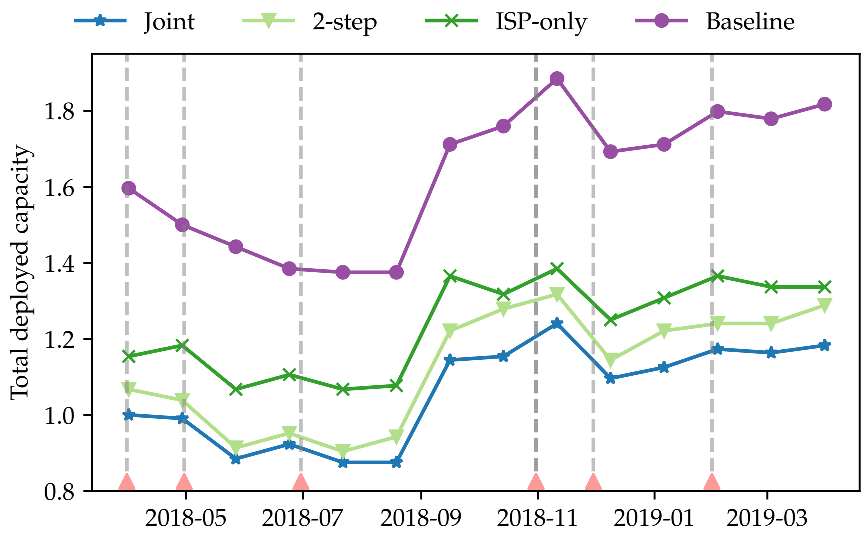


Figure 5.3 Total deployed capacity with reconfigurations on a monthly basis for `HT-only`. Red triangles indicate events at which the peering of at least one of the *hyper-giants* changed. `Joint` optimization consistently deploys the least capacity.

backbone network. In order to create the input for the optimization, the flows are aggregated. `HT` is aggregated based on the *hyper-giant*, the origin peering router, and destination core router of the flows (as IP node of the end-users). Background traffic (`BT`) does not belong to any of the top-10 *hyper-giants*. Here, flows are aggregated based on their origin and destination core IP routers. In addition, the demands are split and aggregated in the temporal domain. As a first step, the average rate per demand and hour is calculated. Finally, the maximum value of all hourly averages in a time window, e.g., two hours or one day, is used as demand value in the optimization. Section 5.4.2 provides more details on the chosen time windows. The evaluation uses either `HT-only` or `AllTraffic`, the combination of `HT` and `BT`, as input to the optimization. The scenario that only optimizes `BT` is out of scope of this study.

Optimization approaches. The main performance objective is the deployed capacity in the IP topology which serves as a proxy for OPEX (e.g., power consumption) and CAPEX (e.g., transceivers to buy). The evaluation examines the optimization approaches from Section 5.3.3 and a theoretical lower bound (*Theo. min.*). The lower bound is the deployed capacity for the case where all end-user nodes are directly connected to only one peering node, building a star topology. Such a star topology requires that there are PoPs with sufficient capacity where all *hyper-giants* peer with the ISP. An assumption that does not generally hold as the set of peering locations usually differs among the *hyper-giants* [13]. Moreover, *Theo. min.* ignores capacity limits of the fibers and number of transceivers per router.

If not stated otherwise, each optimization determines all components of the solution (end-user mapping, IP topology and IP routing). We use the Python API of IBM ILOG CPLEX 12.9 [280] to implement the MIP. The solver time limit is set to 1 hour.

5.4.2 Reconfiguration Intervals

The evaluation starts by analyzing the impact of the reconfiguration interval on the performance relation between the proposed joint optimization and the baselines.

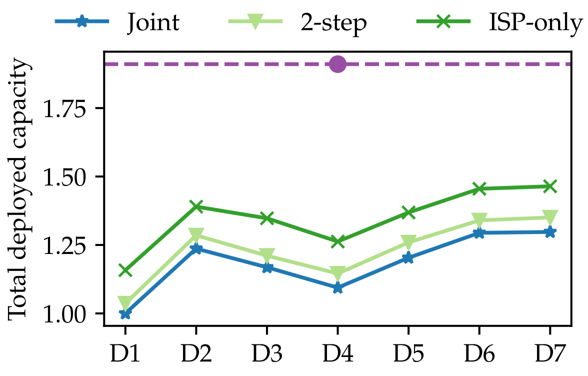


Figure 5.4 Total deployed capacity with reconfigurations on daily basis for `HT-only`.

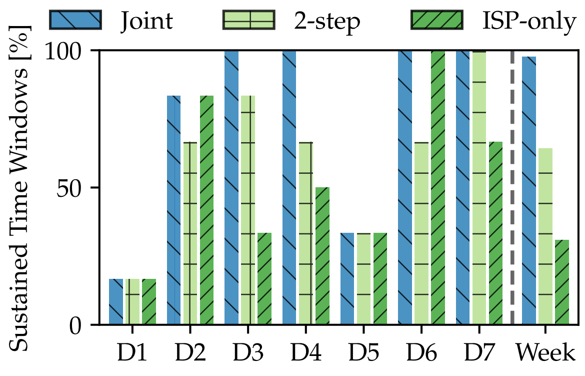


Figure 5.5 Sustained (off-peak) time windows when re-optimizing once per day/week with full knowledge about peak hour. Bars indicate fraction of time windows where the day's/week's peak hour solution is feasible.

5.4.2.1 Monthly-based reconfigurations

According to a previous study, the collaboration between CDNs and ISPs is carried out on a daily to monthly basis [13]. Hence, investigating reconfigurations on a monthly basis is a good starting point. In this case, the re-optimization is performed once per month on maximum values of the weekly peak-hour demands of that month. We investigate the traffic from April 2018 to April 2019 and focus on `HT-only`.

Figure 5.3 shows the total capacity normalized to the value of `Joint` at March 2018. The gray vertical lines indicate dates when the peering of at least one of the *hyper-giants* changed, i.e., peering capacity increased or decreased, or its location changed. Here, we might see whether re-optimization benefits both *hyper-giants* and ISPs.

Reconfiguration leads to capacity savings. For all algorithms, the trend of the total capacity increases over time. `Baseline` performs significantly worse ($\approx$ 35%) on average than all other algorithms. It is followed by `ISP-only` and `2-step`. `Joint` performs best. `ISP-only` changes IP link capacities more aggressively and adapts the IP routing in contrast to `Baseline`. This flexibility already results in the observed performance gain. Adding another layer of reconfigurability, i.e., jointly including the end-user mapping into the optimization, `Joint` saves up to 15% in comparison to `ISP-only`. `2-step`, which performs a greedy mapping, saves only $5-10\%$. This comparison reveals the benefits of joint optimization. Inspecting the evolution of the deployed capacity over time, we observe no clear trend of benefiting from peering changes for all approaches. In summary, introducing monthly reconfigurations leads to substantial savings in infrastructure upgrades. However, the question remains whether such optimization can really sustain all traffic changes within a month: for instance, events can lead to severe traffic changes at specific locations, which might be obscured by the aggregation with the global maximum. We look into this by considering the traffic demands of a week.

5.4.2.2 Weekly-based and daily-based reconfigurations

In order to make the optimization tractable, a week is divided into four-hour-long time windows. The maximum demand hour within a time window serves as input to the optimization. For the

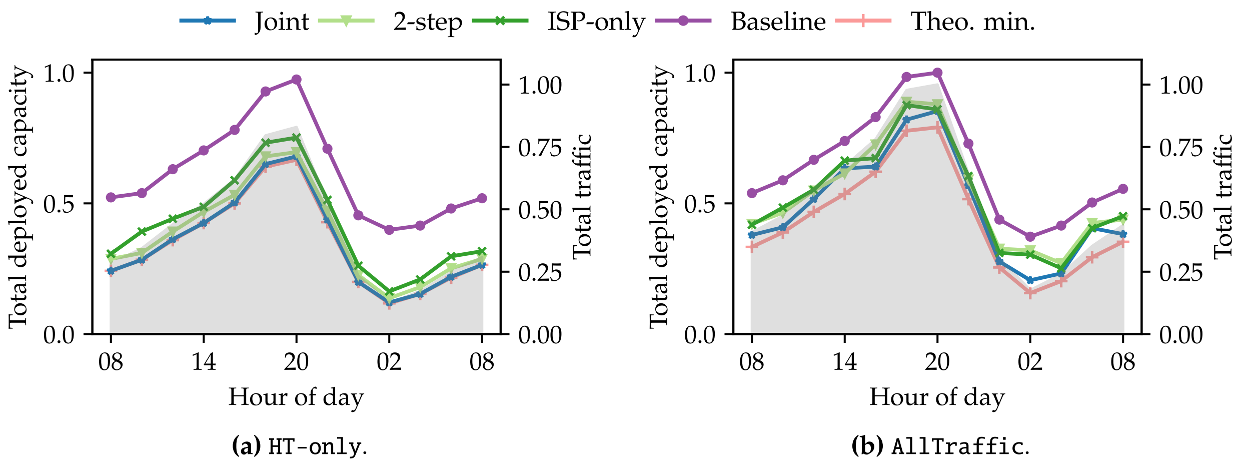


Figure 5.6 Total deployed capacity for re-optimization on bi-hour-scale. Shaded area indicates total traffic volume normalized to maximum value from `AllTraffic`. Capacity savings due to re-optimization throughout the day are observable. `Joint` deploys the least capacity.

initial optimization, we select the peak hour per day/week from these four-hour windows. This assumes that knowledge of the peak hour can be retrieved in advance.

Figure 5.4 illustrates the total deployed capacity when running the optimization for the per-day peak hours. The values are normalized with `Joint` on D1. We observe a similar relationship for the three algorithms as for the monthly reconfigurations. All three optimization algorithms perform better than `Baseline`, for which only the result on D4 is shown as reference. Again, `Joint` consistently outperforms `2-step` and `ISP-only` by 3% and 12% respectively.

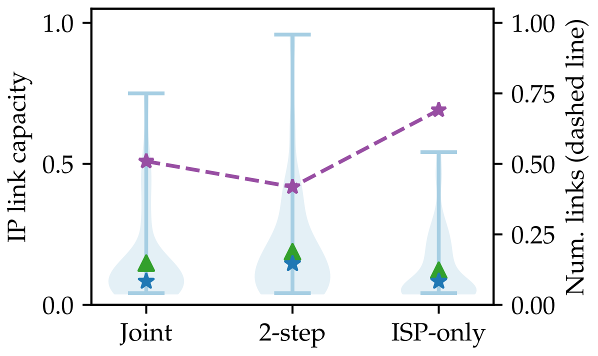
To answer whether the peak hour's solution sustains all traffic patterns of the respective day or week, we run three optimizations per obtained day or week peak hour demand: `Joint`, `2-step`, `ISP-only`. We then evaluate whether the obtained solutions can also serve the traffic in the off-peak hours. Specifically, the MIP's feasibility with fixed variables (with values from the solutions) is checked. If the outcome is positive, the peak hour solution can sustain the traffic, otherwise not. Figure 5.5 visualizes the fraction of time windows sustained by the peak hour solutions.

Coarse reconfiguration intervals do not sustain traffic. The overall result, however, is negative: for all three week-peak hour-based solutions (group on the right), the fractions are $< 100\%$, i.e., there is at least one point in time when the solution cannot satisfy the end-user demands. Similarly, the results for daily re-optimizations are shown. The result is again negative. There are several days of the week for which daily re-optimization is not sufficient. This demonstrates the need for reconfigurations on smaller time-scales.

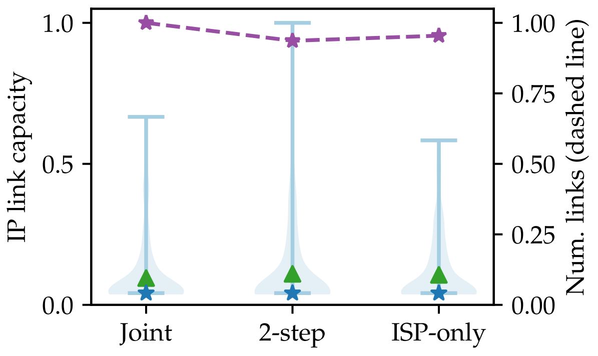
5.4.2.3 Hourly-based reconfigurations

Figure 5.6 shows the total deployed capacity throughout a day for `HT-only` and `AllTraffic`. Optimization is performed periodically every two hours based on the traffic maximum of the considered time-window. The gray area contrasts the total traffic volume. The lines and markers indicate the values of the best integer solutions found within the time-limit of the solver. The values are normalized to the maximum values from the `AllTraffic` case, i.e., maximum value of `Baseline` for capacity values and maximum value of traffic volume for total traffic.

Intra-day reconfigurations save up 44% of capacity over time. As before, all three algorithms save consistently more than 30% of capacity in comparison to `Baseline`. Moreover, they have



(a) `HT-only`



(b) `AllTraffic`

Figure 5.7 Violinplots showing the distribution of normalized IP link capacities and normalized number of links (dashed line).

a consistent order throughout the day: `Joint` performs best with an average gap of 3% to *Theo. min.* followed by `2-step` and `ISP-only` with gaps of 11% and 25%. Thus, jointly optimizing ingress PoPs, routing, and topology can save non-negligible fractions of capacity.

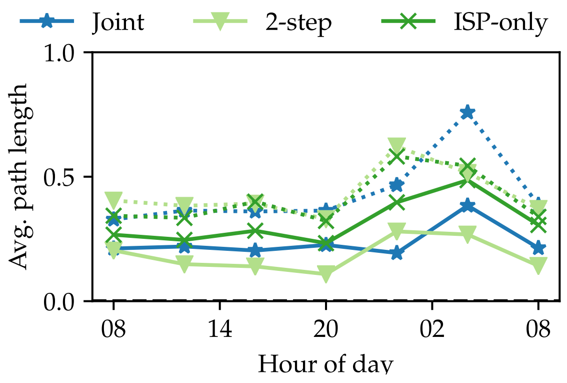
In addition to the savings compared to `Baseline`, reconfiguration on a bi-hourly-bases also opens the possibility to exploit the diurnal demand patterns: The ratio between peak hour (around 20:00) and the deepest valley (around 02:00) is ≈ 7 for both traffic cases. This variance is also reflected in the deployed capacity for all algorithms. Re-optimizing the network saves 44 % of the summed capacity over the day. This directly translates to reduced OPEX, e.g., by energy savings, or freed fiber capacity that can be re-used for other services. Hence, there is a need for reconfiguration on an hourly-basis in order to efficiently operate the network.

5.4.3 Need for Joint Optimization and Reconfigurations

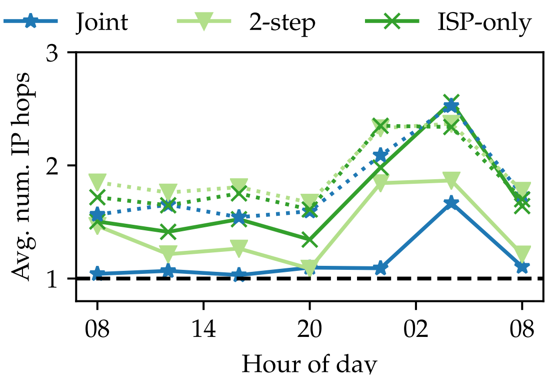
This section answers the question of how joint re-optimization of reconfigurations on multiple layers (the IP topology, routing and end-user mapping) can serve ISPs and *hyper-giants*. Further, it provides insights into the costs in terms of reconfigurations. This section considers both `HT-only` and `AllTraffic`, i.e., it always highlights the differences between both traffic types.

5.4.3.1 The benefits of the ISP and the *hyper-giants*

Whereas the previous section showed that we can generally save capacity when considering *hyper-giant* traffic, this section takes a deeper look into the impact also on the `BT`. `BT` accounts for ≈ 25% of the traffic volume in the ISP infrastructure (cf. Figure 5.6a vs. Figure 5.6b). `Baseline` does not experience a significant rise in the total deployed capacity. The differences are in the range of $3-9\%$. The other approaches, account for this increase in traffic. The deployed capacity increases on average by 45% for `Joint`, 48% for `2-step` and 30% for `ISP-only` when comparing `HT-only` and `AllTraffic`. In particular for the off peak hour, the deployed capacity increases by more than 80% for all of the algorithms. Moreover, the differences between them diminish: `Joint` still achieves the best performance, but `2-step` and `ISP-only` come closer. Nevertheless, re-optimization still exhibits 15 % (in contrast to 30% for `HT-only`) better performance than `Baseline`. We conclude that reconfigurations can still lead to significant savings for ISPs.



(a) Avg. path length.



(b) Avg. num. IP hops.

Figure 5.8 Comparison of average path length (normalized distance) in arbitrary distance unit and average number of hops for `HT` (solid) and `BT` (dashed). Differences between the algorithms vary depending on the hour of the day.

To deepen the analysis of the differences among the optimization flavors, we now look at the attributes of the solutions. Specifically, the following part analyzes their link capacities, the average path lengths, and the average number of IP hops.

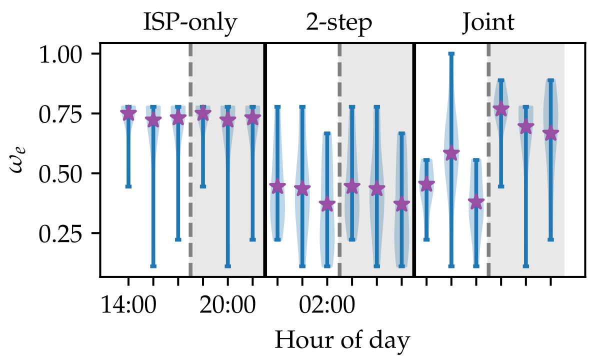
Joint optimization creates smaller topologies. Figure 5.7 compares the number of IP links and the distribution of their capacities for all algorithms with `HT-only` and `AllTraffic`. The figure shows violinplots of the distributions for the sample at 18:00. The markers indicate the mean (triangle) and median value (star). All datapoints are normalized by the largest occurring value across both traffic scenarios.

For `HT-only`, significant differences between the algorithms are visible: `2-step` deploys the smallest number of links followed by `Joint` and `ISP-only`. But it has significantly larger link sizes (the maximum capacity is almost twice as that of `ISP-only`). Moreover, `ISP-only` has the largest number of very small links as indicated by the shape of the violin. This is inefficient from the capacity point of view as it results in large excess capacities for the increase in connectivity. `Joint` is between these two.

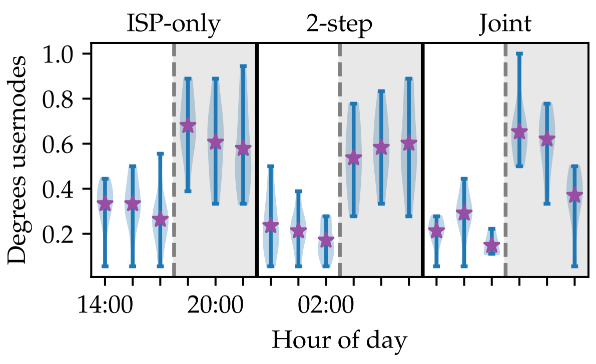
For `AllTraffic`, the differences in numbers reduce, as for all algorithms the number of links significantly increases. However, the medians of the capacity distributions (star) drop for all algorithms which means that the added links have mainly small capacities; they add the necessary connectivity for `BT`. The reduced number of links for `Joint` results in shorter paths for `HT` as described in the following.

Joint optimization decreases mean path length. One primary goal of *hyper-giants* is an efficient delivery of their traffic towards end users. This does not only manifest in high throughput, but also in small latency values. The average path length measured either in IP hops or real distance (arbitrary unit) are two ways to assess this aspect: for instance, the shorter the paths, the lower the expected latency.

Figure 5.8 contrasts the mean path lengths for the `AllTraffic` scenario using two metrics: the geographical path length and the number of hops in the IP layer. The figure also separates `HT` and `BT`. Figure 5.8a compares the average geographical path length of `HT` and `BT`. The values are normalized to an arbitrary unit. We observe that `HT` is consistently routed via shorter paths (average lengths of 0.3) than `BT` (average lengths of 0.45). The algorithms themselves do not clearly



(a) Used CDN PoPs per end-user node.



(b) Degree per end-user node.

Figure 5.9 Number of used CDN PoPs by each approach (a), degree of IP nodes by each approach (b). Optimization scenarios: `HT-only` (white area) and `AllTraffic` (gray area). `2-step` and `Joint` aggregate the end-user mappings and, thereby, they can reduce the IP node degrees.

impact the average path lengths: for instance, `2-step` has sometimes shorter and sometimes longer paths than `Joint`.

In contrast, when looking at the average number of IP hops (Figure 5.8b), `Joint` shows a superior solution: it always has the least IP hops on average for the `HT`. This reduces the total required capacity in the network (bandwidth tax). Here, `ISP-only` suffers from the end-user mapping strategy; the average path lengths are ≈ 50% higher — a clear benefit of the joint optimization. As elaborated later, `ISP-only` has a very diverse end-user mapping which results in longer path to reduce the deployed capacity. On the other hand, `Joint` is able to determine such an end-user mapping that most traffic flows via direct links. For `BT`, the differences among the algorithms are less strong.

In order to understand how the ”designed” topologies look like, the following part compares the approaches with respect to the used CDN PoPs and the IP node degrees. For the used PoPs, we look at each end-user node and determine the number of PoPs this end-user node receives traffic from. For the degree of the IP nodes, we determine the number of links to other IP nodes.

Aggregated end-user mappings reduce IP node degrees by 10%**.** Figure 5.9a shows violinplots of the fraction of peering locations (CDN PoPs) that are used by the end-user connecting IP nodes. The fraction is calculated regardless of the *hyper-giants* as follows:

$$\omega_e = \frac{\sum_{h\in\mathcal{H}}\sum_{u\in\mathcal{U}_h:i(u)=e}\hat{o}_{u,ph}}{\|\cup_{h\in\mathcal{H}}\mathcal{P}_h\|}. \tag{5.18}$$

The solid lines separate the violins of the three algorithms, and the dashed lines separate `HT-only` (white background) and `AllTraffic` (gray background). For each such subgroup of violins, the figure shows a medium load (14:00, left violin), the peak (20:00, middle violin) and valley (02:00, right violin) times of the traffic. Note that the labels on the abscissa are redacted for readability. The three times shown for each algorithm and traffic are the same. The purple star indicates the mean values.

`ISP-only` uses the end-user mappings from the ISP’s data. It shows that most end-user facing PoPs (IP nodes) have *hyper-giant* demands routed via almost all CDN PoPs (≈ 70% on average and in the median), i.e., demands are less aggregated. The behavior is consistent for `HT-only` and `AllTraffic`. For `2-step` and `Joint`, the fractions of used PoPs are smaller on average and have higher variance. The rational behind this is that both algorithms try to group the demands and

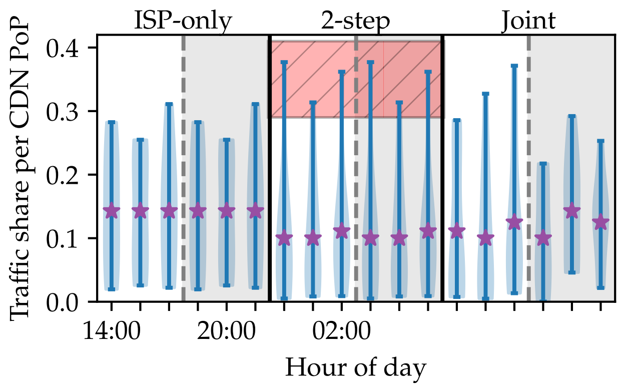


Figure 5.10 Traffic distributions over peering locations. Optimization scenarios: `HT-only` (white area) and `AllTraffic` (gray area). `2-step` performs the most aggregation of traffic to a single PoP.

route them via the same peering PoP and path in the IP topology to reduce fragmentation of IP capacity. Whereas the behavior of `2-step` is again consistent between `HT-only` and `AllTraffic`, `Joint` shows differences between the two traffic scenarios and some variance throughout the day. As the size of the demands increases towards the peak hour and peering capacities are limited, mappings have to be adapted, become more distributed and more PoPs are used during the peak hour. For `AllTraffic`, `Joint` shows similar values as `ISP-only`. As observed before, this scenario results in more links to provide connectivity for the `BT`. Hence, aggregating demands to few peerings is not as beneficial.

Figure 5.9b illustrates that the previously observed behavior leads to reduced nodal degrees of the end-user nodes. Recalling the ideal situation, where all traffic of the end-user node is routed via a single link, this is the desired behavior and the potential that is exploited by the joint optimization. For `AllTraffic`, the differences in node degree are smaller as also observed for the total deployed capacity (cf. Figure 5.6b).

5.4.3.2 Are there any drawbacks for *hyper-giants*?

The optimization in this chapter focuses on cost reductions for ISPs. Although this leads to shorter distances between *hyper-giants* and their end-users, it might affect the utilization of the resource provisioning of *hyper-giants*. Hence, we look at the traffic share of the *hyper-giants*' PoPs, a potential indicator for the peering utilization. The assumption is that *hyper-giants* rather prefer evenly and constantly utilized peerings (better load distribution and fewer changes over the day).

Utilization of peerings is unbalanced. Figure 5.10 shows the allocated traffic share over the peering locations of the *hyper-giants*. It uses the same grouping and datapoints as Figure 5.9a. For `ISP-only`, the load distribution values are similar for both `HT-only` and `AllTraffic`. This is an expected behavior, as the *hyper-giant* demands and the end-user mapping do not change for both optimization scenarios. However, it proves the operation of the optimization approaches. Similar observations hold for `2-step`. However, the overall traffic share is lower on average (0.1 compared to 0.14 for `ISP-only`) and for the majority of PoPs as the shape of the violins suggests. In order to sustain all demands, `2-step` uses a PoP with a high traffic share, as shown by the high extreme value reaching above a value of 0.3 (highlighted by hatched box). `Baseline` is left out as it has the same results as `ISP-only`.

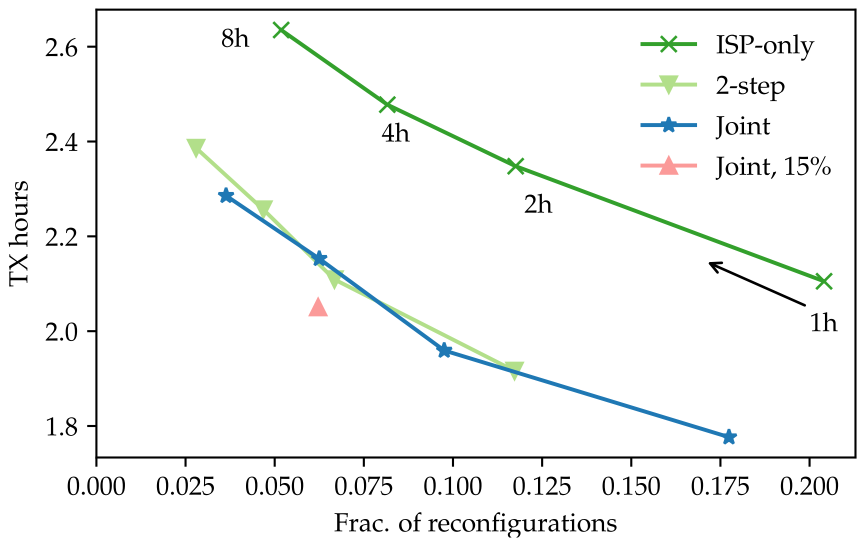


Figure 5.11 Pareto plot of reconfigurations vs. transceiver (TX)-hours (both normalized); traffic is `HT-only`. `Joint` with reconfiguration limit can trade-off the reconfigurations against the deployed capacity.

In contrast to `ISP-only` and `2-step`, `Joint` shows more changes in distribution and variability during the day. Values range from 0.1 to 0.35. Specifically, for `HT-only`, there are some high outliers similar to `2-step`. For `AllTraffic`, the range is smaller. As a consequence, *hyper-giants* might have to adapt their peering capacities, and potentially also their routing towards peering points, over the day more frequently – a potential trade-off for *hyper-giants* between operational efforts and lower latency values.

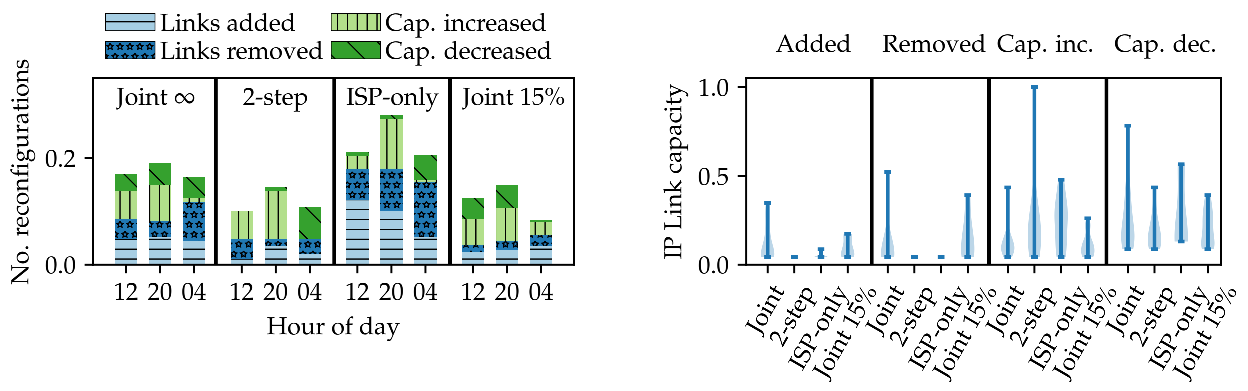
5.4.4 Reconfigurations: Analyzing Topological Impact

So far, the conclusion is positive: joint reconfigurations on multiple layers help to reduce the needed network capacity for the ISPs and shorten the path length from end-users to *hyper-giants*. On the other side, operators have been (and still are) reluctant to use frequent, short-term reconfigurations in their network. Accordingly, this section analyzes (1) the impact of the total number of reconfigurations needed and (2) which type of reconfigurations drive the benefits of joint reconfiguration. Based on such insights, operators can better trade off the costs in terms of reconfigurations and capacity gains according to their preferences.

5.4.4.1 Trade-off between reconfigurations and capacity deployment

As a first simple solution, this part studies the impact of longer reconfiguration periods on the total capacity; the different periods are 1, 2, 4 and 8 hours. As in Section 5.4.2, the optimization uses the peak demands of the time windows.

Large optimization periods reduce reconfigurations. Figure 5.11 shows a Pareto plot between the normalized number of reconfigurations (as defined by the left hand side of Equation 5.16 and accumulated over one day) and the transceiver (TX) hours for `HT-only`. TX hours represent the integral of the deployed capacity over one day, normalized by *Theo. min.* As expected, longer reconfiguration periods reduce the amount of reconfigurations, but they increase TX hours. Among the algorithms, `ISP-only` performs worst in both dimensions. `2-step` results in less reconfigurations compared to `Joint` at the cost of more TX hours which indicates one way to



(a) Number of reconfigurations by type. **(b)** Capacity of changed links.

Figure 5.12 Details on reconfigurations by type, algorithm and affected entities for `HT-only`. `Joint`, which deploys the least capacity in the network, requires more, and more evolved types of reconfigurations.

trade off both objectives. The savings in reconfigurations reduce with increasing periods between re-optimizations.

Considering `Joint`, we observe significant savings (≈ 45%) in reconfigurations when re-optimizing every two hours (0.097) instead of every hour (0.177), while the TX hours increase only by ≈ 10% (from 1.77 for 1 hour periods to 1.95 for 2 hours). For 8 hours, `Joint` has the least reconfigurations but also a significantly higher capacity (increase by 22%). This describes the landscape where operators can trade off the TX hours savings against reconfigurations.

Consequently, an additional constraint *limiting reconfigurations* (cf. Equation 5.16) can make `Joint` a credible alternative for `2-step`. The figure shows that optimizing every 2 hours and limiting the number of reconfigurations to 15 % (triangle pointing up) can provide better results in terms of reconfigurations and TX hours than `2-step`: they are reduced by 3% and 6% respectively. In summary, the proposed model does not only demonstrate the benefit of joint optimization of IP topology, routing, and PoP assignment, but also provides flexibility to adjust results based on, e.g., provider policies. The following part investigates the different types of reconfiguration more in detail.

5.4.4.2 Impact of different reconfiguration types

Figure 5.12 presents a breakdown of the topological reconfigurations into their types: IP links added, IP links removed, IP link capacity increased, and IP link capacity decreased. It compares the three algorithms and `Joint` with 15% reconfigurations allowed for three different times of a day.

Adjacency changes impact only small share of the traffic. Figure 5.12a shows the normalized number of reconfigurations for all types for bi-hourly re-optimization and `HT-only`. While ≈ 50% of the topological reconfigurations for `Joint` and `ISP-only` are adjacency changes, the share is smaller for `2-step`. Also `Joint` in combination with the reconfiguration limit mainly impacts link sizes. Here, the algorithm avoids changes to the adjacency matrix of the IP topology and adaptations are mostly facilitated by changes in capacities. Note, however, that capacity savings cannot be achieved without any link addition or removal. Comparing the three time instances, we observe that the number of reconfigurations peaks at the busy hour of the traffic for all algorithms.

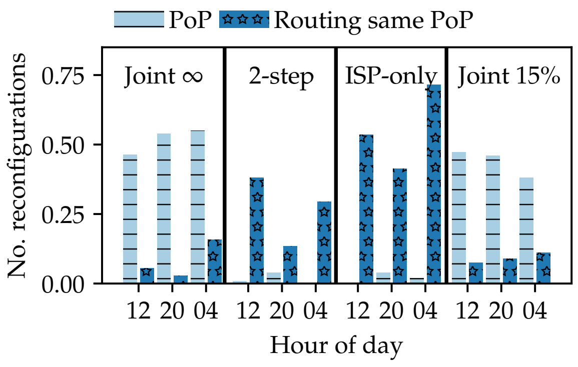


Figure 5.13 Reconfigurations of end-user PoP assignment and IP routing. The behavior of the algorithms varies. `Joint` heavily uses reconfigurations of the end-user mappings to achieve the capacity savings.

Moreover, the share of link removals and capacity decreases is larger for the time instance at 04:00. This seems to be intuitive as the total traffic is low at this time and traffic can be groomed on fewer links.

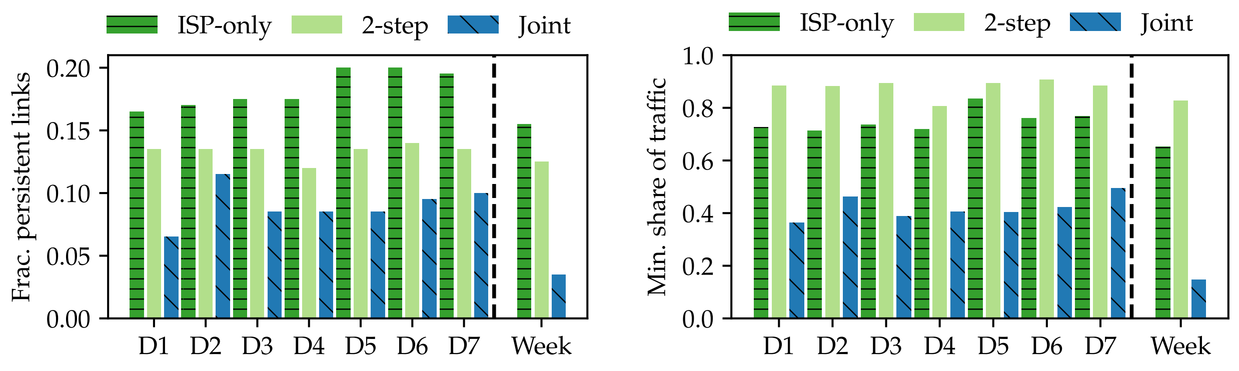
Assessing the size of the modified links, Figure 5.12b illustrates the link capacities before the change (and after the change for additions) for the sample at 20:00. The values are normalized to the observed maximum. While there is no clear pattern among the algorithms for scaling the capacity of links, link additions and removals are restricted to relatively small links. We particularly observe this for `ISP-only` and `2-step` but also to a smaller extend for the `Joint` variants. This suggests, that these changes affect only minor fractions of the traffic, and hints towards a stable topology structure.

End-user re-mappings foster capacity savings. Figure 5.13 visualizes reconfigurations on the higher layers. It shows the changes in the end-user mapping and, in the case that the CDN PoP for a demand does not change, the IP routing. Again the numbers are normalized with respect to the total number of demands, i.e., the maximum number of possible reconfigurations. Three of the four compared algorithms reconfigure either the end-user mapping or the IP routing for at least 25% of the demands. The only exception is `2-step`. It has the smallest number of reconfigurations 13% at 20:00. Comparing the timestamps no clear pattern is evident. The observed benefits in Section 5.4.3.1 results in high numbers of end-user mapping reconfigurations and according routing changes for the demands for `Joint` without (∞) and with limit (15%) on the reconfigurations which represent another factor to trade-off for the reduced capacity. `2-step` and `ISP-only` show almost static end-user mappings but apply many changes in the routing of the demands. Given the superior performance of `Joint`, we conclude that reconfigurations of the end-user mapping foster the capacity savings.

5.4.4.3 Stability of topology

Motivated by the observation that the re-optimization mainly adds and removes low capacity links (cf. Figure 5.12b), this section examines to what extent there is a persistent structure in the topologies. Therefore, it defines a link to be persistent if the link exists throughout multiple optimization periods.

Large fraction of traffic is routed via persistent links. Figure 5.14 visualizes the fraction of links (Figure 5.14a) and their carried traffic (Figure 5.14b) for persistent parts of the topology. The carried traffic is given by the minimum share of traffic that the link carried in the time window.



(a) Fraction of link occurrences.

(b) Share of allocated traffic.

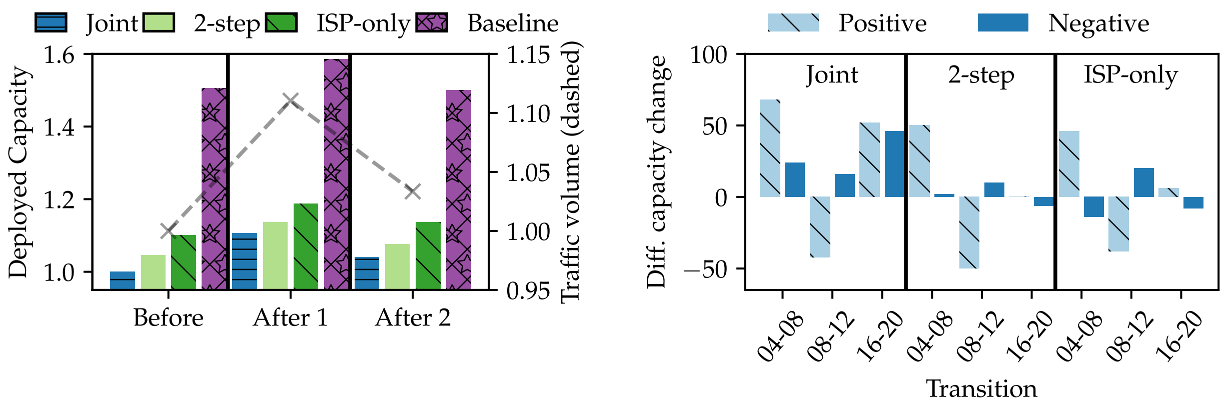
Figure 5.14 Comparison of the stability of the topology. (a) compares the occurrence of unique links over one day and one week. (b) shows the share of traffic over persistent links. All approaches rely on few stable links that carry a large amount of the traffic.

The figures evaluate two time windows for persistence: per day and per week. For `2-step` and `ISP-only`, the fraction of adjacencies that exists over the whole day is consistently $> 10\,\%$ and carries between $70-90\%$ of traffic. For `Joint`, the lower stability in the links on a day-by-day basis ($5-10\%$ of the links) which aligns with the observations of reconfigurations of larger links. Also the fraction of carried traffic over this stable topologies is smaller but with 40% still significant. Considering the span of a week, for `Joint`, less than 5% of the links exist all the time. For `2-step` and `ISP-only`, the fractions are still $> 10\%$ and the links carry $> 60\%$ of the traffic. These observations underline that most reconfigurations affect only small fractions of the traffic and that a significant share of the traffic is routed via stable parts of the topology that are only scaled up and down. Moreover, setting a stable topology on a daily basis and temporarily adding small links upon short term spikes, allows maintaining higher operational confidence while gaining from reconfigurability.

5.4.5 Special Event COVID-19

So far, this study considers "normal" traffic conditions. As a first step to assess the robustness of the presented approach, this section evaluates the benefits of jointly reconfiguring multiple layers in situations with increased and changed traffic patterns. As a case study, it uses data collected during the COVID-19 pandemic. The samples are aggregated from 4 hour windows on three different days. Before is a day in the week just before the major lock-down in the country of the ISP, After 1 and After 2 are the same weekday in the two subsequent weeks.

Figure 5.15a shows the deployed capacity during peak traffic hour for the case of unlimited reconfigurations. The values are normalized with `Joint` and Before. For all algorithms, the deployed capacity increases for After 1 and After 2 in comparison to Before by $\approx 13\%$ and $\approx 4\%$ respectively. This is in line with the increase in traffic volume observed as indicated by the dashed line and also confirmed by other work [254, 281, 32]. The decrease of deployed capacity and traffic volume for After 2 aligns with the decrease of video streaming quality by two *hyper-giants* [282]. The already observed behaviors of the algorithms from the sections before are not significantly affected by the traffic increase with `Joint` being $\approx 10\%$ better than `ISP-only` and $\approx 2\%$ better than `2-step`.



(a) Deployed capacity during peak hour (normalized to `Joint`- Before).

(b) Diff. total capacity change between After 1 and Before.

Figure 5.15 Impact of traffic during COVID-19 on deployed capacity and reconfiguration behavior. The relation among the algorithms persists but the reconfiguration behavior throughout the day changes in response to the changed demands.

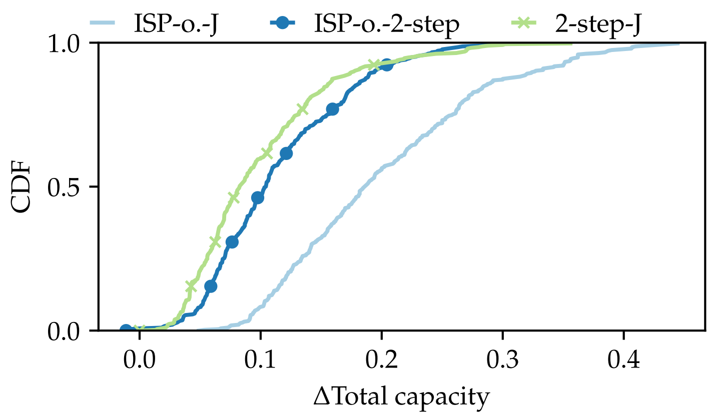
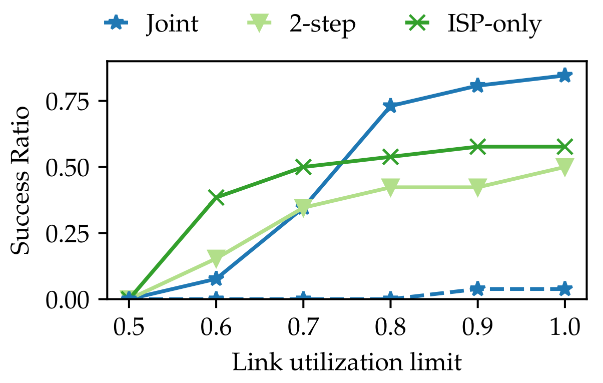
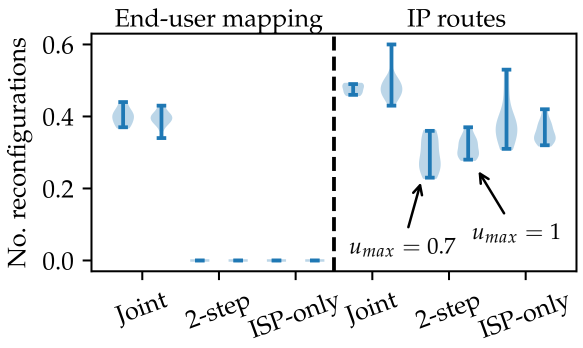


Figure 5.16 Cumulative distribution function (CDF) of relative difference in deployed capacity between optimization flavors for randomized demands. The observed relations between the optimization flavors persist also for the randomized demands.

Events impact reconfiguration behavior. Figure 5.15b contrasts the changes in capacity between After 1 and Before for several transitions throughout the day, namely from 04:00 to 08:00, from 08:00 to 12:00 and from 16:00 to 20:00. The capacity changes are grouped into “positive”, i.e., all link additions and capacity increases, and “negative”, i.e., all link removals and capacity decreases. The figures shows the differences of the summed capacity changes. For all algorithms, After 1 has higher capacity increase from 04:00 to 08:00 compared to Before (difference > 0), while the capacity added in the transition from 08:00 to 12:00 is lower (difference < 0). The differences in negative capacity changes are small for `ISP-only` and `2-step`. Only for `Joint`, we also observe higher differences in the amount of removed capacity. This indicates more changes in connectivity. Thereby, `Joint` can use the ISP’s optical infrastructure and the CDNs’ peerings more efficiently.



(a) Restoration success ratio for single IP link failures against allowed link utilization.



(b) Reconfigurations during restoration.

Figure 5.17 Restoration success ratio (a) and reconfigurations by type (b) for single IP link failures. Traffic is `HT-only`. `Joint` can restore up to 80% of single IP link failures but relies on reconfigurations of both end-user mapping and IP routing. The dashed line in (a) shows `Joint` with fixed end-user mappings.

5.4.6 Randomized Demand Patterns

As a second step to evaluate the robustness of the joint reconfiguration on multiple layers, this section assesses the generalization of the results to randomized demand patterns. As described in Section 5.4.1, the input data has a specific structure, and hence, it is challenging to generate randomized demands that are feasible. For instance, given the peering capacities of the *hyper-giants*, uniform sampling of demands can lead to situations where demands exceed the available capacity of the peerings, or where a feasible routing cannot be found. Therefore, to avoid infeasible problem instances, the existing input data is scrambled along the spatial dimension. A preprocessing step shuffles the assignment of IP end-user nodes and peering nodes to the optical nodes. Thereby, the relations between peering capacities and demand sizes are preserved. This evaluation uses the input data from Figure 5.6a and randomizes it with 30 seeds – 360 samples in total.

In order to account for the high variance in the total traffic volume, Figure 5.16 shows the relative differences between the algorithms by means of the cumulative distribution functions. For instance, “ISP-o.-J” is the difference between `ISP-only` and `Joint`.

`Joint` (“ISP-o.-J”) and `2-step` (“ISP-o.-2-step”) both perform better than `ISP-only`. For `Joint`, all differences are > 0 with an average of 0.2, i.e., `Joint` improves by 20%. `2-step` gains 11% on average compared to `ISP-only` on the randomized data. Moreover, the difference between `2-step` and `Joint` (“2-step-J”) is ≥ 0 for all samples. In 75% of the instances, `Joint` can save more than 5% of deployed capacity compared to `2-step`, and 40% of the instances show savings $\geq$ 10%. Overall, this demonstrates that the gains of joint re-optimization generalize to less structured demand patterns.

5.4.7 Failures

Although the presented approach does not directly optimize the network for resilience, this section provides an outlook on its ability to restore from single failures out of the 25 largest IP links of the topology. Such a scenario reflects failures of transceivers. To evaluate the worst case scenario, it assumes that 100% of the link capacity becomes unavailable. To start with, restoration possibilities are limited to two layers: IP routing and end-user mapping. That is, when re-running the algorithms for restoration, the IP topology is fixed.

Joint IP routing and user mapping restoration increases robustness upon IP link failures. Figure 5.17a compares the ratios of successful restorations of the three algorithms against the allowed link utilization. While `Joint` shows increasing restoration success with more relaxed link utilization and eventually can restore 80%, `2-step` and `ISP-only` saturate quickly around only 50%. The distribution of the IP link capacities (Figure 5.7) reveals that `ISP-only` deploys more links, but the single links' capacity is smaller. This in turn leaves only little headroom to allocate traffic in case of re-routing. On the other hand, `2-step` deploys few large links, which carry high traffic volumes. Failures of those IP links require high excess capacity for restoration, making it less flexible.

To complement this analysis, the dashed line shows the success ratio for `Joint` with fixed end-user mappings from the pre-failure solution, i.e., only the ISP reacts to the failure. In this case (dashed line), the success ratio is zero for almost up to 80% allowed link utilization. For 90% and 100% only ≈ 4% of the failures can be restored. The reason for this negative result becomes evident from the reconfigurations during restoration (Figure 5.17b). The figure shows violinplots of the normalized number of reconfigured end-user mappings and IP routes if the mapping stays constant. For each pair, the left violin shows $u_{max} = 0.7$ and the right one $u_{max} = 1.0$. We observe two differences for `Joint` compared to `ISP-only` and `2-step`. First, it reconfigures more IP routes. The values are larger for both u_{max}. Second, it makes use of changes in the end-user mapping reconfiguring around 40% of them.

In summary, the study of failures shows mixed results. While most of the failures can be restored when allowing reconfigurations on two layers, limiting reconfigurations to only one layer leads to almost zero success. A more detailed consideration of resilience is left for future work.

5.5 Discussion: Reaping the Potential Benefits

The evaluations based on data from a large ISP reveal a significant potential of a joint optimization and adaptation of and for *hyper-giants*' traffic – for the ISP in terms for reduced costs or increased resource efficiency, and for the *hyper-giants* in terms of reduced latency. This section discusses limitations and avenues on how these benefits may actually be reaped.

5.5.1 Generalization and Scalability

Although they contain an initial assessment of the robustness, the evaluations consider only one ISP topology, a limitation of this study that stems from the lack of publicly available data for other topologies (cf. Section 5.4.6). Nevertheless, the results are expected to generalize to other ISP networks as well since those networks share important properties, at least to those of comparable size. First, similar traffic patterns like the diurnal pattern and dominance of *hyper-giants*' traffic in ISP networks have been observed for other ISPs too [283]. Further, traffic demands increase and patterns might change, as observed during the COVID-19 pandemic [32]. The sensitivity analysis with randomized demands confirms the observations and provides further confidence in the robustness of the results.

A second aspect of the generalization is whether the optimization is tractable on other ISPs. The mathematical modeling results in an NP-complete problem which might lead to run-time problems on larger problem instances. Limiting the solver run-time to 1 hour has provided results close to the optimum in our case but might not be sufficient on other topologies or even faster reconfiguration cycles. Hence, designing more tailored algorithms might be necessary for real deployments. Alternatively, machine learning (ML)-based solutions as e.g., [14, 284] could provide means to speed up the optimization and may be explored in future work.

5.5.2 Business-related Trends

Emerging cooperation. Many inefficiencies related to how ISP networks today deliver CDN traffic, are due to the lack of information. While ISPs have detailed knowledge of their network topology (the two lower layers in Figure 5.1), their knowledge of the demand of content and the distribution flexibilities is usually very limited (upper layer in Figure 5.1); and vice versa for the *hyper-giant* [285, 286, 250]. Several recent studies presented promising approaches to improve information sharing and cooperation [250, 273, 277, 13, 270, 287]. Moreover, it has been shown that major players adopt such approaches for information exchange [13], which can and should be extended to account for topological flexibilities too.
***Hyper-giants* acting as ISPs and vice versa.** In addition to emerging collaborations, we can already note that some *hyper-giants* become ISPs and provide connectivity for end users. Either on traditional fixed line access [288] or via less established interconnects such as satellite links [289]. Similarly, ISPs started to provide content services, e.g., [290, 291]. There is an increasing number of entities unifying ISP and *hyper-giant*. Hence, the approach of joint optimization is becoming more relevant.

5.5.3 Technological Trends

From a technical perspective, the joint optimization on multiple layers largely benefits from the increasing adoption of softwarization and centralized control approaches in ISP networks, e.g., [58]. In particular, there are two major aspects to implement the envisioned system.
Centralized data collection and operation. The joint optimization relies on detailed knowledge of the current state of topology and routing, and particularly on information about the future demands. Systems that implement such data collection in a highly scalable way are already in operation at the ISP, e.g., [13], using widely deployed protocols such as BGP or IS-IS for routing data and NetFlow or IPFIX for collection of demand data. Prediction of future demands based on the collected ones is already possible with deployed systems [57] and fostered by omnipresent advances in machine learning. Also data from the CDNs, e.g., server loads or content availability, is already being collected by CDNs to implement their user-mapping systems [261]. Besides the centralized database, also the control and decision-making has to operate on a global level to leverage the joint optimization. Large ISPs and *hyper-giants* have shown that centralized control entities are feasible already today, cf. [240, 241, 58]. Depending on the particular situation, i.e., CDN-ISP cooperation or implemented by one party, detailed integration of all parts into one system is necessary. However, further work to integrate control over the three layers is still necessary.
Deploying configurations. Finally, configurations have to be communicated to the network equipment in case of IP topology and routing or to the mapping systems for content requests.

Run-time reconfigurations of network devices have strongly been fostered with the continuing trend of softwarization and programmability of networks. Most devices offer programmable interfaces to change configurations and the state of the network. For instance, in the optical domain, NetConf and TL1 are prominent examples for such interfaces. But often the specifics are vendor-dependent potentially posing a challenge for system integration. Note that while such interfaces offer easy triggering of changes, actual realization by the device might still be limited by other technological aspects. For the deployment of user mappings, approaches depend on the involved entities. For CDN-ISP collaboration, communication between the parties should happen in a reliable and automated way. ALTO [265] has already been adopted by some players to achieve this [13]. Other approaches might include custom APIs between mapping system and centralized control, e.g., in the case of converged roles of CDN and ISP. Reconfigurations can lead to interruptions or additional overhead, e.g., if states have to be synchronized over the network or if IGP has to re-converge. A specific detail here is the scheduling of reconfigurations to reduce service interruptions and to guarantee correctness of the networking state during reconfigurations, e.g., using a *make-before-break* strategy [292]. The specific costs depend on the actual system design. A deeper analysis particularly for reconfiguring on all three layers is subject to future work.

5.6 Summary

This chapter studies the benefits of joint optimization of reconfigurations on multiple layers, the second characteristic of reconfigurable topologies. Motivated by emerging reconfigurable network infrastructures in CDN-ISP environments, it presents an optimization framework to improve the delivery of *hyper-giant* traffic across ISP networks, leveraging short-term reconfigurability along three dimensions: IP topology, routing, and end-user mapping. Using extensive empirical analyses in collaboration with a large ISP, it finds that such joint optimization is indeed worthwhile, and can benefit both the ISP and the *hyper-giant*. The observed benefits of considering reconfigurations multiple layers are two-fold. First, it reduces required backbone capacity by 15% and second, it shortens path lengths by 30%, both on average and during the critical peak hour. The assessment of the reconfigurations indeed demonstrated that these benefits stem from leveraging the flexibility on multiple layers, and how operators can trade off reconfigurations and deployment cost savings. Moreover, the evaluation shows that infrastructure re-optimizations can be leveraged in specific situations too, for example during the COVID-19 pandemic or under link failures.

This chapter is a first step to render flexible topologies not only demand- but also reconfiguration-aware along the considered networking layers. Such layers might include, the demand allocation, the IP routing and the IP (logical) topology. Besides, the approach opens several interesting directions for future research. First, the current approach assumes a fixed grid optical transport network. A natural extension is to model elastic optical networks (EONs). In addition, while a more efficient use of the physical infrastructure benefits OPEX (e.g., in terms of fewer deployed lightpaths and reduced power consumption), it will be interesting to further evaluate the benefits for quality-of-experience of specific applications. Finally, while the chapter focuses on a large ISP, it would also be interesting to study whether similar benefits can be obtained even in smaller ISPs, and at higher reconfiguration granularity.

Chapter 6

Designing High-Throughput RDCN with Local Routing and Control

We have already observed in Chapter 3 that there are various reconfigurable datacenter networks (RDCNs) in prior work that come with overheads and limitations. A comparison of existing RDCN architectures reveals that they are restricted in their routing, require complex buffering mechanisms, and come with non-standard requirements for transport layer protocols. Specifically, communication on the (dynamic) optical topology is often limited to one or two hops. This constrains the possible path diversity, and hence capacity, of the optical network [207, 47, 53, 66]. Furthermore, a large fraction of the existing RDCNs features a *static* topology part to provide a basic connectivity between the nodes, e.g., [47, 51, 191]. In such topologies, routing is usually *segregated*: flows are either only forwarded along the static or the dynamic topology, but not a combination of both [47, 51]. The restriction to segregated routing also entails overheads as it requires significant buffering while the reconfigurable links are not available.

In order to close this gap, this chapter introduces and evaluates Duo, a high-throughput RDCN architecture. Duo realizes seamless routing between static and dynamic links and, thereby, provides work-conserving forwarding. That is, a normal store-and-forward operation is possible, packets are processed upon arrival, and no external (time-based) trigger, such as in RotorNet [48], is needed. This approach has the potential to avoid long buffering times and hence delays: if a reconfigurable link is unavailable, packets can directly be forwarded to the other available (static) links. However, going beyond segregated and 1- or 2-hop routing requires a novel control plane: traditional routing protocols based on shortest paths are not designed for highly dynamic topologies, and the frequent recomputation of routes can become (computationally) infeasible [293]. Furthermore, to keep update costs low and provide high scalability, it is desirable to have small forwarding tables. To achieve this, Duo uses a de Bruijn-graph-based topology, in which static links are enhanced with opportunistic links. This particular structure allows local and greedy integrated routing, i.e., forwarding rules only depend on local information. It operates a receiver-based approach for the efficient detection of elephant flows as well as the local and collision-free scheduling of demand-aware (*DA*) links. This can significantly reduce control plane overheads during topological reconfiguration, maintaining a simple routing, buffering, and control, and is hence well-suited for highly dynamic networks. Moreover, the de Bruijn structure can be implemented with IP-based logical addressing, and Duo does not need any specialized transport layer protocol. Due to its simplicity, Duo is well-suited to be implemented, e.g., using the Sirius [53] architecture, which was originally designed to be demand-oblivious (*DO*), but can

Table 6.1 Recent (R)DCN designs and their properties. (S: standard; NS: non-standard). Red shaded cells indicate deficiencies in the properties. All existing designs have deficiencies in at least one of the desired properties. (Note that the term *work conserving* has to be interpreted in a store-and-forwarding sense, i.e., this property indicates if external (time-based) triggers are required to start packet transmission.)

		Xpander [211]	Sirius [53], Opera [49]	ProjecToR [47]	Gemini [66]	Duo
Topological Re-configurability	Demand-aware	No	No	Yes	Yes	**Yes**
	Update rate	None	Fast	Fast	Slow	**Fast**
Network Layer	Integrated Multi-hop	Yes	2-hops	1 hop	2-hops	**Yes**
	Work conserving	Yes	No	No	Yes	**Yes**
	Routing mechanism	IP	NS	IP	IP	**IP**
Transport Layer	Congestion control	TCP	NS	TCP	TCP	**TCP**

potentially support *DA* link scheduling. Its control plane can be realized using centralized or distributed algorithms.

This chapter evaluates Duo using extensive packet-level simulations and empirically finds that Duo provides a higher throughput compared to state-of-the-art static and dynamic, networks. Furthermore, Duo's properties allow us to use IP & transport control protocol (TCP) out-of-the-box, without the need to develop new network or transport protocols. We further report on a proof-of-concept implementation of the control and data plane of Duo that demonstrates the feasibility of Duo with standard network stacks. The implementation builds around a single P4 switch, which we use to emulate a scenario with 16 top-of-racks (ToRs).

Content and Outline This chapter is based on and re-uses contents from a previous journal publication [2]. The claim, the theorem and the corresponding proof sketches in Section 6.2.1.3 as well as the specific implementation of the sketch-based large flow detection (Section 6.4.2.2) are contributions of the co-authors but introduced in this chapter for completeness.

In the following, this chapter first compares Duo against related work (Section 6.1). Using the ToR-Matching-ToR (TMT) model, the chapter describes the details of the topology structure, the forwarding approach, and the *DA* link scheduling (Section 6.2). Finally, the chapter evaluates Duo using extensive packet-level simulations covering multiple traffic patterns and compares Duo against state-of-the-art datacenter networks (DCNs) (Section 6.3). In order to demonstrate the feasibility of Duo, this section finishes with a proof of concept implementation (Section 6.4) and is summarized in Section 6.6. The implementation of Duo has been made publicly available at `https://github.com/tum-lkn/duo-simulator`.

6.1 Related Work

This section summarizes important properties of recent DCN architectures in Table 6.1 and, thereby, elaborates the motivation for Duo. The second part gives a short overview of prior applications of de Bruijn graphs.

6.1.1 Putting Duo into Perspective to RDCNs

Table 6.1 shows three categories of properties of DCNs: topological reconfigurability, network layer, and transport layer. For topological reconfigurability, the table indicates if the system uses a *DA* topology and the supported update rate of the topology (unless it is static). Regarding the network layer, it indicates (1) if the routing is fully integrated multi-hop (IMH) or only 1 or 2 hops, (2) if the packet scheduling is work conserving, and (3) if the routing can support (legacy) IP. The term *work conserving* refers to the ability to use standard store-and-forward implementations (as available in commercially off-the-shelf (COTS) switches and network interface cards (NICs)). *Non-work conserving* RDCNs require external (time-based) triggers to initiate forwarding decisions and packet transmission, e.g., as in RotorNet [48] or Sirius[53]. Finally, regarding the transport layer, the table shows if the design supports standard (S), e.g., TCP or NDP, or non-standard (NS) congestion control. The table does not compare the *performance* of the different designs but their main design choices.

The first example is Xpander [211], a state-of-the-art *static and demand-oblivious* topology for DCN, which is based on expander graphs. It supports routing over multiple ($>$ 2) hops and, due to its static nature, provides work-conserving forwarding using IP and standard congestion control. While Xpander has many attractive properties, according to recent results (including those in this section), RDCNs are expected to provide better performance, and particularly higher throughput [53, 294].

The second representatives are Sirius [53] and Opera [49], recent proposals of *dynamic and demand-oblivious* designs for RDCNs. Such topologies are very effective with respect to throughput and flow completion time, but still have several potential deficits. First and foremost, they do not feature *DA* links: while the exact role and use of *DA* topology components in future datacenters (DCs) is generally still subject to ongoing discussions, empirical studies show that demand-awareness can improve throughput under today's typically skewed workload distributions [191, 66]. Furthermore, current dynamic and *DO* designs are limited to at most 2-hop routing on dynamic links and are not work-conserving. There are also open questions regarding the complexity of the control plane, the routing scheme, and the transport layer of these systems.

Next, we consider *dynamic and demand-aware* systems. Systems like ProjecToR [47] use a combination of *DA* optical and electric switches, but do not support IMH routing (ProjecToR uses only 1-hop on *DA* links), are not work-conserving, and their control complexity is not fully determined. Gemini [66] and [64], two recent proposals by authors from Google, make a case for *DA* links in production-level datacenters. Both implement only infrequent topology updates (about once a day). They provide work-conserving routing that supports IP but are limited to 2-hop routing.[1]

Summary: Unlike all the above systems, Duo features all the desired properties listed in Table 6.1. It supports IMH routing as Xpander, uses *demand-aware* links as ProjecToR, is *work-conserving* as Gemini, enables *fast topology updates* as in Opera and Sirius, and is based on simple control and IP-based routing. Its properties, especially work-conservation, allow us to use standard TCP in Duo, as demonstrated by the packet-level simulation and the prototype implementation.

[1]Gemini optimizes using a global view of the network which requires a centralized collection of the network's state and hinders high frequency topology updates. Moreover, its topology and routing updates might affect multiple devices across the whole network adding complexity to the control plane and further limitations to the update rate. Gemini operates on pod-to-pod level, while Duo is a ToR-to-ToR level topology. Lastly, not all algorithmic details are available. So a closer comparison is not possible.

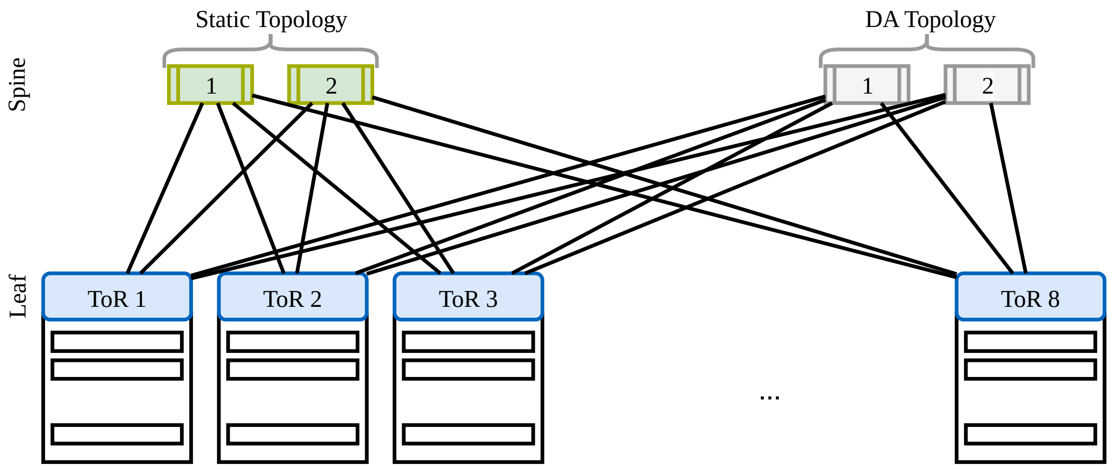


Figure 6.1 TMT network example for Duo with eight ToR switches and four spine switches. Each spine switch has eight input and output ports. Two of the spine switches are static matchings and two are dynamic matchings, which can be reconfigured in a *DA* manner.

6.1.2 Applications of de Bruijn graphs

De Bruijn graphs have been applied in networking before. The first example is distributed hash tables like Koorde [295] that are used in peer-to-peer networks. Koorde improves upon the general design of Chord [296] by introducing a routing that requires $O(\log n)$ hops to reach a node. Besides, one major stream of work revolves around the design of networks-on-chips. For instance, Hosseinabady et al. [297] propose such an architecture. They compare it to mesh and torus architectures and stress the low energy consumption of their proposal as one of the main advantages. Another example is given by Chen et al. [298], who present a 3-dimensional network design and demonstrate its superiority over mesh-based architectures. A second stream considers the design of reliable, static networks for wide area networks (WANs) and DCs. Examples are, among others, [299, 300, 301]. The presented topologies are evaluated with respect to their throughput and delay performance. The work closest to Duo is [302]. The author presents a de Bruijn-graph-based static DCN. He demonstrates how greedy forwarding can be translated to longest prefix match (LPM) forwarding rules. The performance is evaluated using emulations and packet-level simulations and compares several routing schemes, showing a higher throughput of the de Bruijn-based solution.

Summary: In contrast to the works above, this thesis is the first study to show the benefits of a de Bruijn graph in the context of reconfigurable and *DA* datacenter networks.

6.2 The Duo RDCN Design

Duo bases on the TMT model from Section 3.6. In particular, Duo is hybrid. The first part of the topology is static and *DO*, using k_s static spine switches (i.e., static matchings). The second part is dynamic and *DA*, using k_{da} reconfigurable spine switches (i.e., dynamic matchings), and $k = k_s + k_{da}$. The number of *DO* switches is $k_{do} = 0$.

Figure 6.1, illustrates an embodiment of the TMT model for Duo with $n = 8$ ToR switches and $k = 3$ spine switches, from which $k_s = 2$ are static and $k_{da} = 1$ is dynamic. Each ToR-spine link in the figure represents one directed uplink and one directed downlink. It is important to

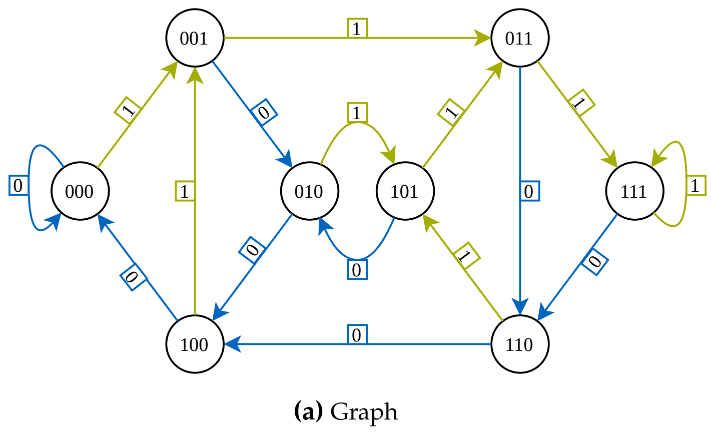


Figure 6.2 Example of a $DB(2,3)$ static & directed de Bruijn graph with eight ToRs and its two corresponding matchings (colored in green and blue). Each port (edge) is labeled 0 or 1 according to the performed shift operation.

note that abstractly, the model uses k spine switches, each implementing an $n \times n$ matching, but each matching can be split across a set of several smaller switches, like in Sirius [53] where spine switches have $\sqrt{n}$ ports.[2]

In order to maximize performance, Duo uses dynamic, *DA* links to provide shorter paths for elephant flows, while other flows are transmitted via the combined (static + dynamic) topology. A key feature of Duo is that it supports *integrated multi-hop* routing across *both* switch types. This is different to previous works that rely on *segregated* and single-hop forwarding for *DA* links [47, 45]. A namely exception is *cp-Switch* which, however, assumes that capacities of static (electrical) and dynamic (optical) links are different [128].

Moreover, the routing in Duo is *efficient*. It relies on logical addressing and a local control plane to implement a greedy routing strategy. Thus, links can always be used immediately, with a work-conserving scheduler. To detect elephant flows, Duo leverages a simple sketch, sampling the flow sizes and then adjusting the dynamic links accordingly. The following sections present the different components of Duo in detail.

6.2.1 The Hybrid Topology

The two topology components of Duo form an augmented de Bruijn network [303]:

- **Static and demand-oblivious de Bruijn topology (backbone):** The static topology of Duo relies on a de Bruijn graph. It is formed by k_s static optical circuit switches (OCSs) or patch panels.
- **Dynamic and demand-aware topology:** The static topology is enhanced by k_{da} reconfigurable matchings, also implemented with OCSs. The demand-aware (*DA*) links add *shortcuts* on top of the static de Bruijn topology.

We first discuss the static de Bruijn topology. The choice of the de Bruijn topology is based on two main properties: (1) It can be built from a small number of matchings (already two matchings suffice in contrast to, e.g., hypercubic topologies that require $\log n$ matchings), and (2) it enables

[2]Note that a smaller number of ports might affect the performance but this is left for future study.

IP-based greedy forwarding also when additional shortcuts are added to the topology, which in turn enables fast and low-overhead topology reconfigurations.[3]

6.2.1.1 The Static de Bruijn Topology

We start with formally defining the de Bruijn topology [304]. For $i \in \mathbb{N}$, let $[i] = \{0, 1, \ldots, i\}$.

Definition 1 (de Bruijn topology)**.** *For integers $b, d > 1$, the b-ary* de Bruijn graph of dimension *d, $DB(b, d)$, is a* directed *graph $G = (V, E)$ with $n = |V| = b^d$ nodes and $m = |E| = b^{d+1}$ directed edges. The node set V is defined as $V = \{v \in [b-1]^d\}$, i.e., $v = (v_1, \ldots, v_d), v_i \in [b-1]$, and the directed edge set E is:*

$$\{v, w\} \in E \Leftrightarrow w \in \{(v_2, \ldots, v_d, x) : \ x \in [b-1]\} \tag{6.1}$$

Note that the directed neighbors of node v are determined by a left *shift* operation on the address of v and entering a new symbol $x \in [b-1]$ as the right most (least significant) symbol. The de Bruijn topology has the following properties [304]:

1. Considering self-loops, $DB(b, d)$ is a b-regular directed graph.
2. Each de Bruijn graph of dimension d is the line graph of $DB(b, d-1)$.
3. Each de Bruijn graph is Eulerian and Hamiltonian.
4. $DB(b, d)$ supports greedy routing with paths of length at most d.

As a consequence of Property (1) above and Hall's theorem [307], we obtain the following observation:

Observation 1. *A $DB(b, d)$ topology can be constructed from the union of b directed perfect matchings.*[4]

It essentially means that Duo can be build using the TMT. Figure 6.2 illustrates the $DB(2, 3)$ de Bruijn topology with $8 = 2^3$ nodes (ToRs) and two matchings (colored in blue and green). Each node in the topology has two outgoing and two incoming directed links (including self-loops). The figure shows the *labeled* version of the graph where each edge (or outgoing port) is labeled with 0 or 1 according to the shift operation implied by Equation 6.1.

It follows from Observation 1 that we can build a $DB(k_s, d)$ topology with k_s static spine switches. From Property 4, it follows that when n is a perfect power of two, we can build a topology for n ToRs and with a diameter $\log n$ with $k_s = 2$.

6.2.1.2 Greedy and LPM Routing in the de Bruijn Topology

Due to its structure, the de Bruijn topology supports greedy routing from a source s to a destination t based solely on the address of t. That is, to choose the next-hop toward t, each node on the route needs to know the address of t and the addresses of its neighbors. The next-hop is chosen as the neighbor which minimizes the *de Bruijn distance* to t. The de Bruijn distance between

[3]In principle, other approaches that enable greedy routing are possible alternatives to the de Bruijn topology like hypercube and Butterfly networks [304], small world networks [305] or distributed hash tables networks such as Chord [296] or Kademlia [306]. The de Bruijn topology provides the best tradeoff between their low degree, the network diameter and IP integration. Future work may consider other alternatives.

[4]A perfect matching of a graph is a matching in which every vertex is incident to exactly one matching edge.

Algorithm 2 Building the DB Forwarding Table

```
   Function BuildTable (at node $v$)
1: for each neighbor $z_1 z_2, z_3$ at port $p$ do
2:     Add the following entries to the table
3: Reduce the forwarding table according to LPM rules
```

Prefix	Port	Path-length
$z_3 * *$	p	3
$z_2 z_3 *$	p	2
$z_1 z_2 z_3$	p	1

Prefix	Port	Path-len
0 ∗ ∗	0	3
10∗	0	2
110	0	1

(a) Neighbor 110 on port 0

Prefix	Port	Path-len
1 ∗ ∗	1	3
11∗	1	2
111	1	1

(b) Neighbor 111 on port 1

Prefix	Port	Path-len
0 ∗ ∗	0	3
10∗	0	2
110	0	1
111	1	1
011	Local	0

(c) Reduced Table for ToR 011

Figure 6.3 The results of building the forwarding table (Algorithm 2) of ToR 011 with neighbors 110 and 111 in the static $DB(2,3)$ de Bruijn graph.

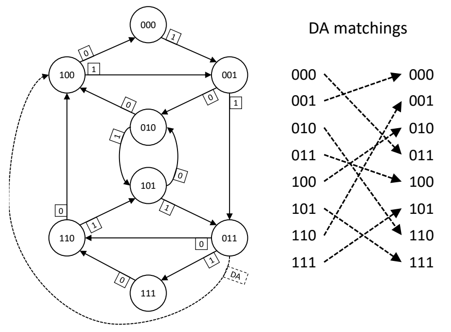
two nodes v, w, denoted as $\text{dist}_{\text{DB}}(v,w)$, is the minimum number of *shift* operations needed to transform v's address to w's address, i.e., the length of the shortest path. The main observation is that each such shift implies a directed edge and the next-hop in the routing. Considering the example in Figure 6.2, the de Bruijn distance between nodes $s = 011$ and $t = 001$ is $\text{dist}_{\text{DB}}(s,t) = 3$. The route from s to t in $DB(2,3)$ is $011 \to 110 \to 100 \to 001$. Note that each hop reduces the distance to t by 1.

Routing on the de Bruijn topology can be realized via a simple forwarding table that is based on a *LPM* [302]. This results from the fact that the de Bruijn distance from a node $v = (v_1, \ldots, v_d)$ to *all* other nodes can be compactly represented using d entries and LPM. For example, the de Bruijn distance between v and all nodes whose address has a LPM to rule $(v_3, v_4, \ldots, v_{d-2}, *, *)$ is two. Namely, two shift operations suffice to transform v's address to the address of a node that matches the above rule.

Building the forwarding table for a node v only requires knowledge of the address of each neighbor w and the respective outgoing port p that connects to it. Algorithm 2 describes the generation of a forwarding table (for simplicity only for the $DB(2,3)$ case) and Figure 6.3 shows the forwarding table of node 011 and how it is built from its two neighbors 110 and 111. Note that in this example rules $1 * *$ and $11*$, are removed from the forwarding table of ToR 011 in the *table reduce* process since they will never be used (there will always be a longer matching prefix for these rules).

Considering Algorithm 2, we can state the following about the *size* of the forwarding table of each node in the de Bruijn topology [302]:

Observation 2. *The LPM forwarding table of each node in a $DB(b,d)$ topology has at most $bd = O(b \log_b n)$ entries.*



(a) ToR 011 with new DA link to 100.

Prefix	Port	Len
$0**$	DA	3
$00*$	DA	2
100	DA	1

(b) Entries from 100.

Prefix	Port	Len	IP
$0**$	$\{0, DA\}$	3	10.0.0.0/9
$00*$	DA	2	10.0.0.0/10
$10*$	0	2	10.128.0.0/10
100	DA	1	10.128.0.0/11
110	0	1	10.192.0.0/11
111	1	1	10.224.0.0/11
011	Local	0	10.96.0.0/11

(c) Reduced table on ToR 011.

Figure 6.4 The new forwarding tables of ToR 011 after the establishment of the DA-link from 011 to 100.

Having understood the forwarding in a fully static de Bruijn-based topology, we can now discuss the DA links and how they are merged into the hybrid topology.

6.2.1.3 The Dynamic Demand-aware Topology

The simplicity of Duo relies on the observation that adding shortcuts to the static de Bruijn topology is easy and maintains the ability for greedy and LPM routing. Recall that there are k_{da} reconfigurable switches which means k_{da} matchings of size n for DA links. For now, we consider these $k_{da} \cdot n$ DA links as arbitrary links. Later we discuss how to choose these links based on the demand.

Let $G = DB(b, d)$ be a de Bruijn topology over the node set V. Let M be a directed matching on $V \times V$. Let $H = G \cup M$ be the union of the directed graphs G and M with the same set of nodes V.

Figure 6.4 (a) demonstrates this case and shows H, the union of a $DB(2, 3)$ topology with a single DA matching (showing for readability only one DA link from 011 to 100). Figure 6.4 (b) and (c) present the new forwarding entries for the added DA link and the reduced table at node 011. Similar to the fully static case, the forwarding table is constructed using Algorithm 2 but now considering an additional neighbor from the DA link. If we consider as before the route from $s = 011$ to $t = 001$ it will now be shorter $011 \to 100 \to 001$. In fact, all packets that reach node 011 with destination addresses with LPM $00*$ will use the new DA port for forwarding. Note also that routes toward addresses with LPM $0**$, like 010, have now two equal length routes (of length three), for example, $011 \to 100 \to 001 \to 010$ or $011 \to 110 \to 101 \to 010$. More generally, we can claim the following about H.

Claim 1. *Let H be the union of a de Bruijn $DB(b, d)$ topology and a single directed matching over the same set of nodes. If we perform Algorithm 2 on each node in H, then H supports integrated, multi-hop, and greedy LPM routing with forwarding table size of $(b + 1)d$.*

Proof sketch. First we can show that H supports greedy routing, since in each node v and for each destination t the next-hop will be the *neighbor* of v in H with the shortest *de Bruijn distance* to t. While greedy routing on the static topology $DB(b, d)$ reduces the de Bruijn distance in each hop by exactly one, DA links can reduce it by more than one. So the distance is strictly decreasing in each hop until the destination. From the greedy routing it is clear that LPM forwarding will work and that the path is integrated in a multi-hop manner. The forwarding table size is at most

$(b+1)d$ since each node has at most $(b+1)$ neighbors (b static and one from the DA matching) and for each it needs d entries to represent its de Bruijn distance to destinations.[5] □

Following Claim 1, we can extend the single matching case to more than one matching and support k_{da} DA matchings. Formally, for integers $k_s, k_{da}, x \geq 2$ and $n = (k_s)^x$, we denote by Duo(n, k_s, k_{da}) the Duo topology with $k = k_s + k_{da}$ spine switches, backbone network $DB(k_s, \log_{k_s} n)$, and k_{da} DA switches. We can state the following about the hybrid topology of Duo.

Theorem 1. *The Duo(n, k_s, k_{da}) topology with n ToRs and $k = k_s + k_{da}$ spine switches ($k_{da}, k_s \geq 2$) supports integrated, multi-hop, greedy, LPM routing with forwarding table size of $O(k \log_{k_s} n)$ and diameter $d \leq \log_{k_s} n$.*

Proof sketch. The proof extends Claim 1 to k_{da} demand-aware matchings. Since the additional links can only reduce distances the diameter of Duo(n, k_s, k_{da}) is at most the diameter of $DB(k_s, \log_{k_s} n)$ which is $\log_{k_s} n$.[6] □

6.2.2 Scheduling of Demand-Aware Links

Duo relies on a control plane which can use centralized or decentralized scheduling of the DA links. The centralized scheduling benefits from the global view, while the decentralized scheduling supports fast reaction.

Sirius' [53] reconfiguration model serves as basis for DA links: spine switches use passive gratings while (sending) ToR switches rely on tunable lasers which determine the link to set up in the corresponding switch (matching). This property is useful for the distributed version of the scheduling where the receivers provide permissions to senders to reconfigure links. All algorithms use the command 'Set DA-link (x, y, i)' which means that sender ToR x tunes its laser on (egress) port i to establish a direct link to ToR y via switch i. Recall that each ToR has k egress and k ingress links toward the k spine switches, so egress or ingress port i can be mapped to switch i.

6.2.2.1 Centralized scheduling of *DA*-links

The centralized scheduling uses a greedy heuristic to add shortcuts (i.e., DA links) to the backbone de Bruijn network. It periodically, every *update period* ρ, determines the new DA-links based on an estimate of the accumulated demand or a measurement of the traffic in the network until the current time. This estimation is described by a *demand matrix* D. The algorithm sorts the demands in D by their size in non-increasing order and for each demand $(s, t, d_{s,t})$ in D, it tries to add a DA link to the network. In case the algorithm decides to set a DA link, the link is not available for use during the *reconfiguration delay* r_{da}, unless it was already setup previously. Thus, the (worst-case) slot length is $s = \rho - r_{da}$. This results in a duty cycle of $\epsilon = \frac{s}{s+r_{da}}$. The algorithm is repeated until no new DA link has been added, potentially iterating several times over the list of demands. Moreover, remaining free, i.e., unmatched, ingress and egress ports are matched. If possible, the matchings are in parallel to links of the static de Bruijn topology, otherwise, a random matching is applied.

[5]Claim and proof have been contributed by co-authors of [2]
[6]Theorem and proof have been contributed by co-authors of [2]

Algorithm 3 Centralized (Greedy) DA links setting

```
   Function Greedy-DA-links (D - Demand Matrix, $k_{da}$ - number of DA switches)
1: $\Delta = \{(s, t, d_{s,t}) \in D : d_{s,t} > v_{th}\}$, sorted by volume $d_{s,t}$
2: for all $(s, t, d_{s,t}) \in \Delta$ from large to small do
3:     for all $i \in 1 \dots k_{da}$ do
4:         if $s, t$ have available ports in DA switch $i$ then
5:             Set DA-link $(s, t, i)$ and break loop.
```

Algorithm 4 Distributed DA-link scheduling receiver-side

```
   Function DistDaRx () at destination $t$
1: Upon detection of elephant flow from source $s$
2: if $t$ has available DA ports then
3:     Send PortRequest(ports) to node $s$
4:     if $s$ reply with PortApprove(i) then
5:         DA-link $(s, t, i)$ is set (with timeout)
```

Algorithm 3 shows the centralized algorithm, *Greedy-DA-links*. The algorithm is a simple version of greedy k-matchings (known also as b-matching for undirected graphs [308]). The algorithm iterates over requests $(s, t, d_{s,t}) \in D$ that are larger than a threshold v_{th} in non-increasing order and only connects a direct link between s and t if they have unmatched ports on the same DA switch i. The shortcuts added by Greedy-DA-links support IMH. The following presents a similar version of this algorithm that is easier to implement in a distributed way.

6.2.2.2 Distributed scheduling of *DA*-links

Algorithms 4 and 5 show the distributed scheduling algorithm, *DistDA*, for the receiving and the sending ToR. The algorithms combine similar approaches as presented in ProjecToR [47] and Sirius [53]. They implement a distributed, threshold-based greedy k matchings algorithm. The destination ToR runs an elephant flow detection which in turn triggers Algorithm 4. For instance, the elephant flow detection can be implemented in P4 using sketches [309] as discussed in more details later. If any of the destinations (ToR) detects a source(-ToR) as elephant it checks if it has available DA-ports. If available, the destination sends an offer, *PortRequest($ports$)*, to the elephant source ToR via the static topology part where $ports$ is a list of available ingress ports at the destination. Upon reception, the source/sender checks for an available egress DA-port on its side. If a port is available, it acknowledges the request via a *PortApprove(i)* message and sets the link on port i. If no DA-port is available at the source, the request is declined. The destination ToR continues to generate *PortRequests* for other elephants. An agreed DA-link, i.e., the ports at sending and receiving ToR, is reserved for fixed period of time ρ. Afterwards, the ports can be assigned to new requests and the circuit might be reconfigured. However, the circuit is *not* pro-actively torn down but kept alive until an appropriate request arrives.

While the distributed scheduler is simple, it is effective as we will see next. The study of more sophisticated schedulers (e.g., based on distributed stable matchings [47] or online algorithms [310]) is left for future work.

Algorithm 5 Distributed *DA*-link scheduling sender-side

```
   Function DistDaTx() at source $s$
1: Upon PortRequest(ports) from destination $t$
2: if $s$ has available DA ports in $ports$ then
3:     Send PortApprove(i) to node $t$
4:     Set DA-link $(s, t, i)$ (with timeout)
5: else
6:     Send DeclineRequest
```

6.2.3 Implementation and Practical Aspects

This section discusses some implementation-related and practical aspects of Duo.

6.2.3.1 Implementation and Cost

While not being limited to specific features of it, Duo is envisioned to be implemented using the Sirius architecture [53]. Sirius is captured by the TMT model, but one of its great advantages is that instead of spine switches, Sirius uses a single layer of k arrayed waveguide grating (AWG) routers. AWG routers are simple and *passive* without moving parts and do not consume power (Chapter 3). Still, each grating diffracts ("forwards") incoming light from input to output ports based on the wavelength and thereby, abstractly creates an matching. Reconfiguration is then performed by a physical-layer ToR switch (or directly on servers) equipped with k transceivers containing tunable lasers that can change the wavelength used to carry the data toward the gratings through an optical fiber.

Sirius has been presented as a *DO* architecture which provides fast end-to-end reconfiguration, due to a pre-determined, static schedule that specifies the connectivity at any given fixed-size timeslot. However, the Sirius architecture is in principle also well-suited for *DA* scheduling, with slower end-to-end reconfigurations.

As Duo differs from Sirius only in the scheduling and routing, the cost and power consumption of Duo are comparable to Sirius. In [53], the authors show that Sirius' power and cost are about 25% of an electrically switched Clos network. That said, unfortunately, a direct comparison of the performance of Duo and Sirius is currently not possible as Sirius' simulation code is not available. Therefore, the empirical evaluation concentrates on the comparison to Opera [49] which is similar in the *DO* nature as Sirius. Additionally, the evaluation compares to static expander topologies which are also state-of-the-art datacenter networks [211].

6.2.3.2 IP Addressing and LPM Forwarding

Section 6.2 states that Duo builds around LPM. In fact, the (logical) de Bruijn addresses in Duo can be embedded into the network hosts' IP addresses rendering Duo feasible on available (COTS) hardware. The approach presented in the following uses IPv4 but can also be implemented using IPv6. Depending on the number of ports per ToR, a single symbol of the de Bruijn address takes one or multiple bits of the IP address: $q = \lceil \log_2(b) \rceil$. Thus, the full de Bruijn address occupies $q \cdot d$ bits of the IP address. In order to use LPM to implement the forwarding, the IP address is partitioned into three parts. The first p bits mark the base network that is assigned to Duo. The following $s' = s \cdot d$ bits identify the ToR by means of the de Bruijn address. The remaining h bits identify the host/VM inside the rack, i.e., each ToR is assigned a $/(p + s')$ prefix.

For the example of Figure 6.4(a), the de Bruijn address can directly be mapped to an IP address/prefix and occupies only 3 address bits. Figure 6.4(c) shows an exemplary forwarding table for ToR 5 = 011 with 10.0.0.0/8 as the base IP prefix. In this case, the IP 10.160.32.15 is decomposed as

$$10.160.32.15 = \underbrace{00001010}_{p}.\underbrace{101}_{s'}\underbrace{00000.00100000.00001111}_{h}. \tag{6.2}$$

Following Algorithm 2, each node can build its IP forwarding table locally based on its ToR neighbors' addresses. In particular, when a new *DA* link is established for a node's port and the node knows the ToR address of the new neighbor, the forwarding table can be updated locally (without recomputing the shortest paths).

Transport Layer In general, Duo does not require a customized transport protocol nor complex flow scheduling. It performs well with existing protocols "out of the box", for instance with TCP or NDP [172]. However, handling small and large flows differently, e.g., by using different transport protocols and separating them into different queues, can significantly improve performance further, and reduce flow completion times of the small flows drastically. Thus, similar to previous approaches such as Opera [49], Duo classifies flows according to a fixed threshold f_{th} into *low latency* and *bulk* traffic. In order to improve the performance for low latency flows, NDP [172] was identified as a promising candidate [49]. As we observe later, it also shows good performance in Duo. For large flows ($> f_{th}$), Duo uses TCP for congestion control and re-transmissions.

Each port of a ToR (static and *DA*) in Duo provides three queues of different, fixed priorities. In order to avoid interference on congestion feedback, TCP acknowledgements and NDP header packets (i.e., acknowledgements or packets that have been stripped) are always forwarded via the high priority queue. NDP data packets use the medium priority queue and TCP data packets use the low priority queue. Packets residing in the queues of a *DA*-link that is reconfigured are dropped. This is a radical approach but the imposed re-transmissions have been found to impact Duo's performance only marginally. Forwarding these packets via the new *DA*-link resulted in lower throughput.

6.3 Evaluation

This section uses and extends *htsim* to evaluate Duo. *htsim* is a packet-level simulator which has also been used in previous works, e.g., [49, 172]. With the simulations, this section answers the question of how Duo fares against state-of-the-art topologies such as more common *DA* topologies, expander-based networks, and Opera [49]. It further explores how Duo behaves under varying traffic conditions and assesses the impact of several topology configurations.

6.3.1 Methodology

The following part describes the simulation settings and traffic patterns more in detail.

Settings The evaluation considers topology configurations with $n = 64, 128$, and 256 racks. Each ToR has 16 ports which are divided into 8 downlink ports for the hosts, and 8 uplink ports towards other ToRs. Since Duo implements a directed topology, the uplinks are split into 8 ingress and 8 egress ports. All links have a capacity of 10 Gbps (as in previous work [49]). So the considered topologies have total uplink capacities of 5.12 Tbps, 10.24 Tbps and 20.48 Tbps.

Unless stated differently, the reconfiguration delay is $r_{da} = 100$ µs for *DA* links. This delay includes both computation and link setup. The active period of an *DA* link, i.e., the slot duration, is $s = 9.9$ ms, i.e., the *DA* links are updated every 10 ms (if needed). Overall, this setting has a duty cycle of 99%. The reconfiguration delay for *DO* links in Opera is 10 µs and following the default configuration yields a duty cycle of 98% [49]. The packet size is 1500 B. Unless stated differently, each simulation runs for 10 s of simulation time.

Traffic and Flow Size Distributions The evaluation considers an online traffic scenario in which flows arrive according to a Poisson process. It assesses the following four traffic scenarios:
Permutation. The flow sizes are fixed to 500 kB and the spatial communication pattern follows a rack-to-rack permutation matrix. The permutation matrix was shown to be the worst case traffic for expander and dynamic *DO* (rotor-based) topologies like Opera [294, 3].
Hadoop, Websearch and Datamining. In each case, the flow sizes are sampled from the respective flow size distribution as reported in the literature [193, 311, 34]. They are similar to a power law distribution, but with different average and maximum flow sizes. Consequently, most of the flows are small, but the majority of the traffic belongs to large flows.

The arrival rate is adapted in order to create load values in the range from 10% to 80% with respect to the total uplink capacity of the hosts. A load of 100% means that all hosts are sending at full line rate.

Topologies The evaluation compares Duo to three combinations of topologies and resource management, i.e., *DA* link configuration and routing, algorithms, representing the state-of-the-art.
Duo. Our proposal uses Duo ($k_s = 2$, $k_{da} = 6$) and configures the *DA* links using the centralized greedy, one-hop approach shown in Algorithm 3 with $v_{th} = 10$ MB. For all flows that are present in the system when the algorithm is called, it has knowledge of the remaining volume to be transmitted. Upon establishment of a circuit, it updates the routing information (locally) to incorporate the new *DA* link using Algorithm 2. In addition, it ensures that only equal cost (length) paths according to the greedy de Bruijn routing are used.
Segregated. The first baseline (Segr) has the same topology configuration and *DA* link scheduling algorithm as Duo but can only use *segregated* routing similar to ProjecToR [47]. It routes flows via a *DA* link only if the link directly connects the source and the destination of the flow. Otherwise, flows are routed on the static topology according to the greedy de Bruijn routing.

Note that de Bruijn graphs serve as the backbone topology for this baseline since a good expander with such a small degree ($k = 2$) cannot be constructed and a higher degree expander would result in an unfair comparison as the number of *DA* links is smaller.
Expander. The second baseline (Exp) is a static $k = 8$-regular directed *Expander* topology that is built from k static matchings (similar to [211]). For flow allocation, it uses a k-shortest path routing where flows are randomly assigned one of the k paths upon their arrival.

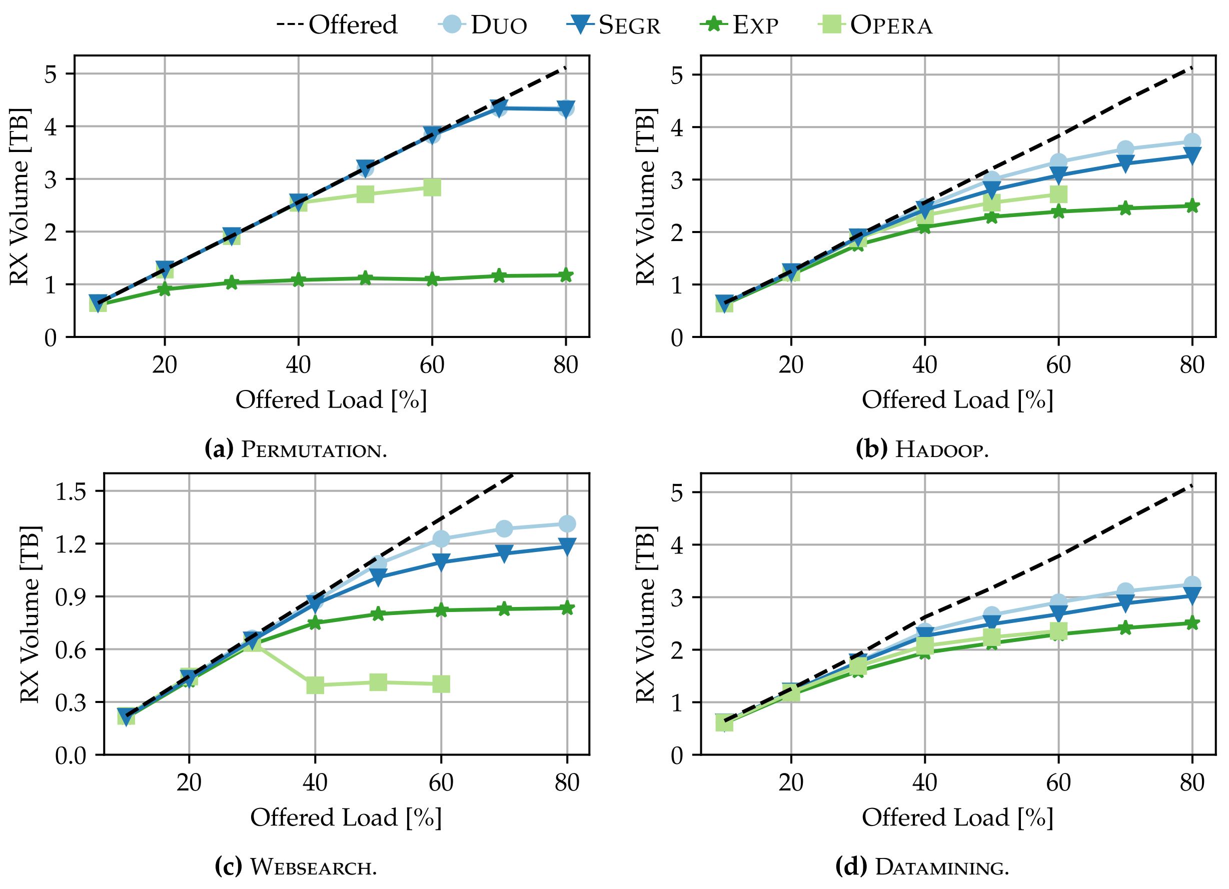


Figure 6.5 Total received volume within 10s against total offered load. The dashed line indicates the offered load. Duo outperforms the baselines Segr, Exp, and Opera.

All three systems use the priority queueing described in Section 6.2.3.2. Each port has a low priority queue which can hold 50 data packets and medium and high priority queues of the same size. Flows $\leq f_{th}$ use the NDP [172] implementation and large flows ($> f_{th}$) use the TCP implementation available in *htsim*. If not stated otherwise, the threshold is f_{th} = 1MB.

Opera. The third baseline is Opera [49], a state-of-the-art dynamic, *DO* topology. Opera periodically cycles through a set of matchings maintaining an expander graph at every time instance. This thesis uses the available implementation in *htsim* and the default configuration as provided with the original paper [49]. Flows < 15MB are sent using NDP [172] via the expander part, whereas large flows use the RotorLB (RLB) transport protocol and rate allocation algorithm. There are two queues per port on the ToRs. The first (low priority) queue can hold 8 data packets and the second (high priority) queue has the same size for header packets. Note that in addition, Opera makes heavy use of packet queueing on the hosts and its transport protocol is not standard.

6.3.2 Empirical Results

The empirical evaluation revolves around the following main *takeaways*:

6.3.2.1 Duo increases system throughput

Figure 6.5 presents the total received (RX) volume throughout one simulation run. It compares Duo, Segr, Opera, and Exp for the four traffic patterns. The abscissa shows the offered data relative

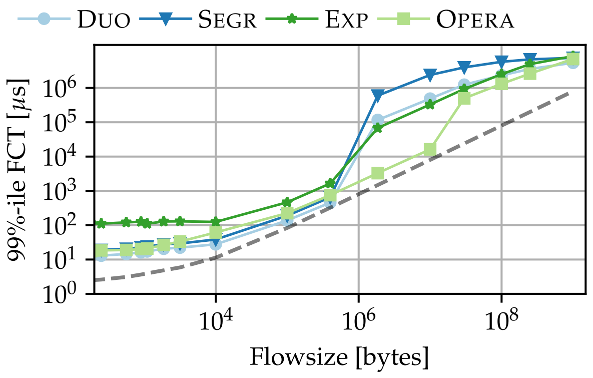


Figure 6.6 cumulative distribution function (CDF) of the 99%-ile of the flow completion time (FCT) by flow size. The relations between the algorithms change depending on the flow size.

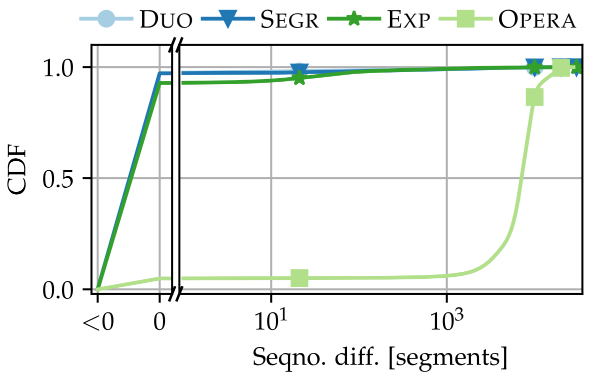


Figure 6.7 CDF of the difference between received and expected segment sequence number as measurement of packet reordering. Duo has less packet reordering than the other solutions.

to the uplink capacity of the ToRs and the simulation duration. The offered data describes the goodput (dashed line), i.e., it does not account for transport layer effects. The RX volume directly correlates with the throughput of the network. Naturally, it increases with the offered data but saturates when the throughput of the network is reached. Recall that Duo, Segr, and Exp use TCP for large flows so the received data is ordered while for Opera the packets mostly arrive out of order due to the oblivious scheduling of the next hop link.

For three of the traffic patterns, Duo improves over all baselines. Only for Permutation (Figure 6.5a), it is on par with Segr. Duo and Segr saturate beyond 70% offered load which corresponds to the ratio of the *DA* link capacity to the total link capacity (= 6/8 = 0.75). In this case, the *DA* links are used to exactly match the traffic pattern, i.e., the permutation matrix. Thereby, the network provides paths of minimal length and serves the traffic efficiently. Opera saturates around 50% which corresponds to the upper bound for the throughput that has previously been derived in theory.[7] Exp starts to saturate already around 20% offered load.

For Hadoop (Figure 6.5b), Websearch[8] (Figure 6.5c) and Datamining (Figure 6.5d), the differences between the topologies are smaller. First, we note that Opera and Exp align and start saturating both around 40% for Hadoop and around 30% for Websearch and Datamining. Moreover, the RX volume of Opera drops for loads > 30% with Websearch. This is in line with the original paper of Opera which also observes decreasing throughput when the load of Websearch traffic increases [49, Figure 10]. The volume of latency sensitive traffic (flows < 15 MB) exceeds the capacity of Opera's expander part.

Second, Duo successfully transmits ≈ 15% more data on average across the load values in comparison to Opera and Exp. For instance, Duo successfully transmits 2.75 TB for 50% offered load from Datamining while Exp achieves only 2.27 TB and Opera 2.24 TB. Compared to Segr, Duo transmits 8% more data for high loads on average across the three workloads. This illustrates the benefit of the IMH routing provided by Duo. Since the results until here are structurally similar for the traffic distributions and for the sake of readability, the remainder of the evaluation focuses on the Datamining workload.

[7]Opera is only evaluated up to 60% load due to computation constraints.
[8]Websearch is simulated only for 3 s due to the significantly larger number of flows.

6.3.2.2 Duo preserves competitive flow completion times

Besides throughput, the FCT is an important performance indicator. Figure 6.6 illustrates the 99%-ile of the FCT per flow size for 40% load. The dashed line indicates the minimum FCT under ideal conditions. For small flows (< 1 MB), DUO slightly outperforms the other topologies. For instance, it achieves FCTs around 13 µs followed by OPERA (18 µs) and SEGR (20 µs). EXP gives the worst FCTs, approximately one order of magnitude larger compared to DUO.

The advantage of DUO persists until f_{th}, i.e., a flow size of 1 MB is reached. Here, OPERA starts to perform better. The curves separate before they converge again for flows > 15 MB. The major reason for these differences is that DUO trades off the lower FCT for small flows against increased FCTs for medium sized flows. The choice of f_{th} to define low latency flows controls this trade-off (cf. Section 6.3.2.5). We note, that DUO consistently performs better than SEGR across all flow sizes. Overall, DUO effectively trades off the higher throughput and slightly better FCTs for small flows against the FCTs of medium flows.

6.3.2.3 Duo has a moderate amount of packet reordering

Figure 6.7 visualizes the difference of the received and the expected sequence number per packet arrival at the destination. The figure shows the empirical cumulative distribution function. The abscissa is broken into two parts. The two data points in the left part (< 0 and 0) denote the fraction of packets where the difference was negative and equal to 0. The right part visualizes the samples > 0 with a log-scaled abscissa. A negative difference means that a packet is a bad (unnecessary) re-transmission. The values are below 0.5% for all systems. The observations vary when the difference is ≥ 0. OPERA receives only a small fraction of around 5% of the packets in correct order (= 0). For 80% of the received packets, the difference of the sequence numbers lies between 4001 and 20k. While this is a direct consequence of RLB's packet indirection, it still illustrates the complexity that OPERA imposes on the receiving hosts. For EXP, ≈ 93% of the packets arrive in order while it is ≈ 97% for DUO and SEGR. In conclusion, in spite of the *DA* link and routing reconfigurations, DUO's does not lead to an increased amount of packet re-ordering.

6.3.2.4 Duo path characteristics

To shed light on how DUO's traffic allocation differs from SEGR, EXP, and OPERA, Figures 6.8–6.10 show several aspects of the resource utilization and routing characteristics. Figure 6.8 shows the (weighted) average path length in hops calculated based on the traffic allocation. It weighs each path length with the fraction of traffic that arrived at the destination and used a path of this length. Lower values are better and indeed we observe that DUO reduces the path length for all evaluated load values compared to EXP and OPERA. However, compared to SEGR, it only provides shorter paths for loads up to 40%. The advantage compared to EXP and OPERA decreases with higher load, when DUO starts to route flows via IMH paths that use both *DA* and static links. For 20%, DUO has an average path length < 1.25 hops. This is slightly smaller than SEGR (1.3 hops) but a 37% reduction compared to OPERA and EXP which are ≥ 2 hops. For 80% load, DUO uses only 1.63 hops on average, still a 15% improvement to EXP and OPERA but 15% larger than SEGR (1.42 hops). However, as we observed before in Figure 6.5d, SEGR does not benefit from the shorter average path lengths.

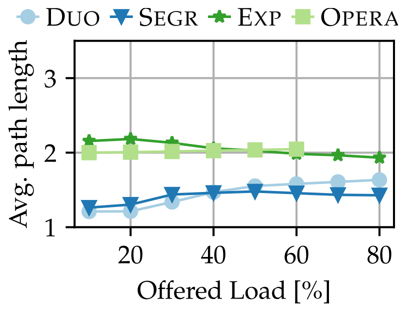


Figure 6.8 Comparison of the traffic weighted average path length against the offered load.

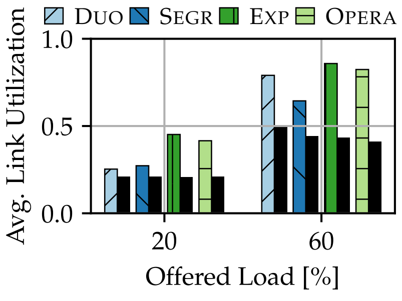


Figure 6.9 Comparison of the average link utilization for two load values.

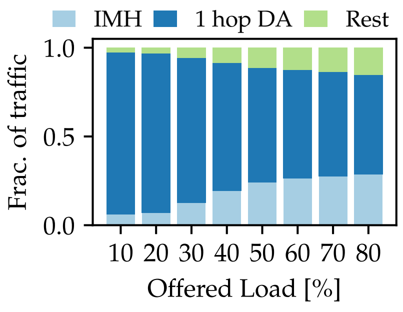


Figure 6.10 Comparison of Duo paths' characteristics against the offered load: IMH vs. 1-hop DA.

To understand this more in detail, Figure 6.9 shows two assessments of the average link utilization from the steady state of a simulation run. The colored bars are the raw average link utilization. The black bar next to a colored bar is the respective *normalized link utilization*. The normalized link utilization $\tilde{u}$ is defined as the ratio between average link utilization u and average path length p, i.e., $\tilde{u} = \frac{u}{p}$. It gives an indication for the throughput.

For 20%, u varies for the networks. Duo has the lowest raw link utilization with 0.25 followed by Segr (0.27), Opera (0.41) and Exp (0.45). This is in line with the order for the average path lengths. The normalized link utilization is around 0.2 for all topologies. This means that all topologies can serve the load as also seen in Figure 6.5d. In this case, a lower link utilization means more efficient operation and hence, is favorable. The observations change for 60% load. Here, Segr has a lower raw link utilization (0.64) than Duo (0.79). The order for Exp (0.85) and Opera (0.82) remains the same. Also the normalized link utilizations behave differently. In line with the observations in Figure 6.5d, Duo has the highest value (0.5) followed by Segr (0.44), Exp (0.43), and Opera (0.41). The lower link utilization of Segr is not an indicator for more efficient transmission here. Since Segr is restricted to 1-*DA* hop paths, it is not able to utilize the available capacity as effectively as Duo. Moreover, it routes more flows via the static de Bruijn topology which increases congestion. As a result, Segr provides lower throughput (cf. Figure 6.5d). This confirms that Duo sets *DA* links that are usable by the greedy routing.

Figure 6.10 provides more details on the path characteristics. It shows the fraction of traffic that uses a single hop, *DA* link (1 hop DA) and the fraction of traffic that traverses *integrated multi-hop* (IMH) paths, i.e., paths which use both static and *DA* links, so their length is at least two hops. With higher load, the share of 1 hop DA traffic decreases, while the fraction of IMH paths increases. This can be explained with the properties of the traffic. For low load, the *DA* links can support most of the traffic, but as load increases more ToR pairs have demand for shortcuts, and multi-hop paths are needed to enable higher throughput. For these paths, due to its greedy routing, Duo also can use the *DA* links and the share of IMH traffic increases. Specifically, we observe rather large values for the fraction of 1-*DA* hop traffic (> 0.9) for loads below 40%. It decreases to $\approx$ 0.56 with 80% offered load. In contrast, the share of IMH traffic starts < 0.1 and increases towards 0.28 for high load. This demonstrates the importance of the IMH paths property of Duo. Without this property, the throughput would be lower, as the results for Segr show (cf. Figure 6.5).

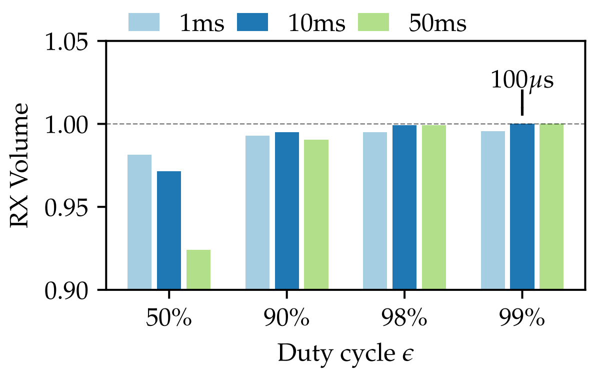


Figure 6.11 Impact of period ρ and duty cycle on total RX volume. Load is 40%. The duty cycle is the major influencing factor on the RX volume.

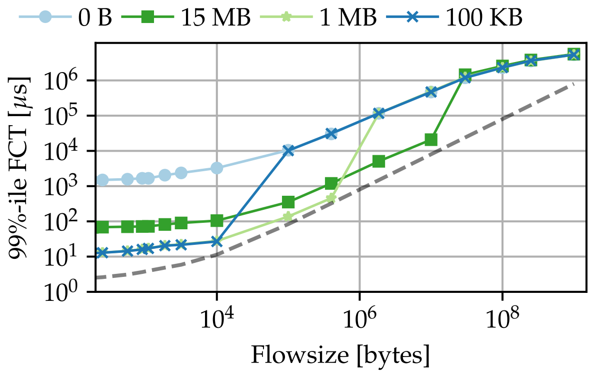


Figure 6.12 Impact of the small flow threshold f_{th} on the 99%-ile of the FCTs by flow size. 0 B classifies all flows as large. Besides the flows whose classification changes, a higher threshold also increases the FCT of the small flows.

6.3.2.5 Sensitivity analysis

Next, the evaluation focuses on how DUO's performance depends on the algorithm configuration. Figure 6.11 approaches this question on two different dimensions: the optimization period ρ ($= s + \delta$) and the duty cycle ϵ. The values are normalized to the result for $\rho = 10$ ms, $\delta = 100$ µs which gives $\epsilon = 99\%$ and is used in the rest of the analysis.

Overall, the impact of the configuration is small. The values range from 0.92 to 1.0. The received volume increases with the duty cycle since less time is spent during reconfigurations. For instance, $\rho = 1$ ms increases from 0.98 at $\epsilon = 50\%$ to 0.99 with $\epsilon \geq 98\%$. The $\rho = 50$ ms shows the largest gain due to an increase of the duty cycle. Here the normalized RX volumes grows from 0.92 to 1.0. Finally, we note that the relation between the three optimization periods is moderated by the duty cycle. While $\rho = 1$ ms is dominating for the duty cycle $\epsilon = 50\%$ with $\rho = 50$ ms being the worst, the opposite is the case for $\epsilon = 99\%$; this effect can mainly be deduced to the absolute value of the reconfiguration delay with low duty cycles.

Another parameter is the flow classification threshold f_{th}. Its impact on the total RX volume is $< 0.1\%$ except for $f_{th} = 15$ MB which has 3% less RX volume. Figure 6.12 illustrates the impact of f_{th} on the 99%-ile of the FCTs for 40% offered load. We observe the impact of f_{th} in two aspects: 1) using the medium and high priority queues for small flows improves their FCT. For instance, flows of size 180 B have FCTs of $\approx$ 1.4 ms without priority queuing ($f_{th} = 0$ B). The values are reduced with $f_{th} = 15$ MB to $\approx$ 67 µs and to $\approx$ 12 µs with 100 kB and 1 MB. Reducing f_{th}, i.e., considering fewer flows as small, results in lower FCTs for small flows. 2) the "jump" of the FCT values moves as expected. Flows that are classified as large have around two orders of magnitudes larger FCTs compared to their classification as small flows.

6.3.2.6 Duo has low complexity

Figure 6.13 illustrates the wall clock time for the simulations against the load. Intuitively, the run-times increase with the load. However, DUO, SEGR, and EXP have a slower increase in run-time compared to OPERA which frequently runs a costly rate allocation.

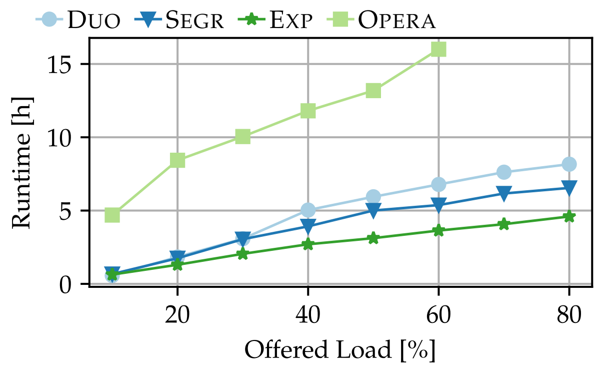


Figure 6.13 Simulation wall clock time against the offered load. Duo has lower runtimes than Opera which hints at lower complexity.

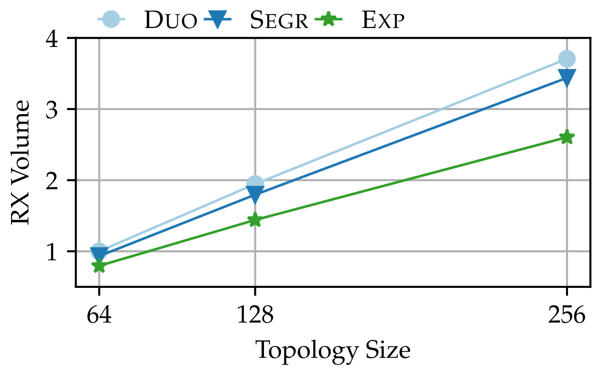
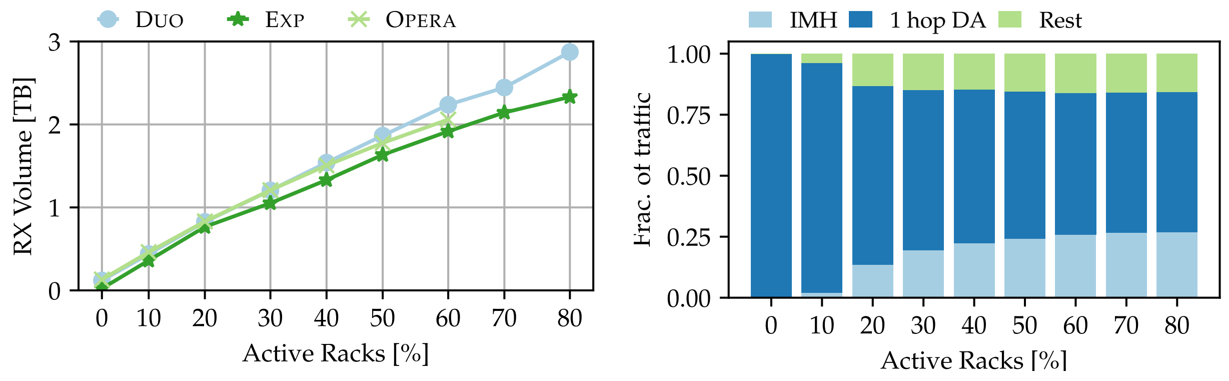


Figure 6.14 Total RX volume against the topology size. The observed gains in RX volume of Duo agaisnt Segr and Exp exist also in larger topology settings.



(a) Received Volume. **(b)** Path characteristics.

Figure 6.15 Total RX volume and fraction of traffic forwarded via IMH paths for skewed connectivity patterns. The observed gains and behavior from the uniform pattern persist also in skewed settings.

We acknowledge that the simulation run-time does not reflect hardware implementation characteristics but is a first indicator for the computational effort and the complexity of the load balancing protocol (RLB) in Opera.

6.3.2.7 Duo's performance gains persist for larger topologies

This section evaluates Duo on topologies with $n = 128$ and $n = 256$ racks. The traffic is generated based on the Datamining distribution and offers a load of 60% relative to the topology size. Figure 6.14 illustrates the RX volume after 5 s of traffic and simulation. The results are normalized to Duo with $n = 64$. The simulations runs are shorter than 10 s due to scalability. We focus only on Duo, Segr and Exp in this comparison since Opera's simulations do not scale well. All topologies have $k = 8$ uplinks. Comparing the RX volume across the topology sizes, the difference between Duo and Exp increases from 0.21 with 64 racks to 1.1 with 256 racks. For Segr, it increases from 0.07 to 0.27 respectively. This result validates that Duo can provide a higher capacity topology also at larger scales.

6.3.2.8 Performance under skewed connectivity patterns

The previous evaluation of Duo covered only uniform connectivity patterns. In order to complement the analysis, we now consider skewed connectivity patterns in which only a fraction of the ToRs communicate. The communicating (active) ToRs are selected randomly from all ToRs and a uniform communication pattern is used inside this subset of ToRs. On average, each of the active racks generates traffic at full rate using flow sizes from the Datamining distribution (similar to [211]).

Figure 6.15 shows the total RX volume for Duo, Exp and Opera as well as the path characteristics for Duo. For the total RX volume (Figure 6.15a), we first observe that Exp performs worse than Opera and Duo for all levels of skewness (percentages of active racks). Second, Opera performs similar as Duo until the total load in the networks approaches the performance limits of Opera that were observed previously, i.e., 50% active racks. At this point, Duo starts to transmit more volume than Opera.

Taking the path characteristics into account (Figure 6.15b, similar representation as in Section 6.3.2.4), we note that strongly skewed communication patterns result in only little IMH traffic as the amount of *DA*-links suffices to serve almost the whole traffic via 1 hop *DA* paths. With the pattern becoming more uniform, the share of IMH traffic grows. In conclusion, Duo performs similar as the baselines in skewed communication patterns but is able to outperform them as the pattern becomes more uniform as it can utilize IMH paths.

6.4 Proof-of-Concept

To demonstrate feasibility, this section presents and evaluates a proof-of-concept implementation of Duo. It starts by describing the testbed (Section 6.4.1) and detailing the implementation of data and control plane (Section 6.4.2). Finally, it presents measurement results from two traffic patterns (Section 6.4.3).

6.4.1 Testbed

We first introduce the components and structure of the testbed and then, provide details on the packet forwarding to emulate multiple ToRs.

6.4.1.1 Components and structure

The proof-of-concept uses a WEDGE 100BF-32QS Barefoot Tofino 3.2 Tbps switch and the P4 language to implement Duo's data plane logic and to emulate a 16 ToR scenario. Figure 6.16 overviews the testbed (the figure shows an 8 ToR version for readability). The switch (gray box) connects to four servers (blue boxes) with 128 GB of RAM and Intel Xeon Silver 4114 @ 2.2 GHz (20 cores). All servers run Ubuntu 18.04 (5.15.0-47-generic kernel). Each server emulates four servers in four racks (green boxes). Therefore, it has four NICs that connect via a 4x10G breakout cable to the switch. For each rack, it generates and sends traffic via one of the four links. The logical separation is realized using *Linux network namespaces*. The Tofino switch emulates the data plane of the ToR switches and the spine layer with its static and dynamic matchings. The round boxes inside the switch indicate the QSFP ports; the white squares inside show the individual SFP+ lanes. Eight loopback cables (double headed arrows) represent the links between different

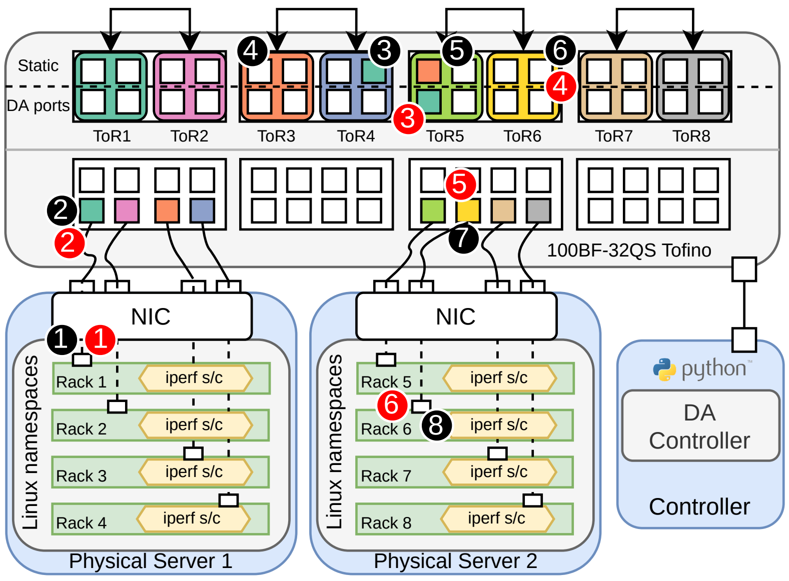


Figure 6.16 Conceptual overview of Duo's proof-of-concept implementation in an eight ToR testbed setup. The shown setup uses two servers, one switch with Tofino ASIC and one dedicated controller machine. Note that the measurements later are conducted with 16 ToRs, so use twice the number of servers and ports at the Tofino.

switching elements; the QSFP ports are operated in breakout mode (4x10G). The lower row of QSFP ports connects to the servers, whereas the upper row is used to create the matchings on the spine switches. The "upper" row of ports on the spine layer corresponds to Static ports, the lower row is used for the dynamic matchings. In the following, the term "port" refers to a single SFP+ (10G) lane.

The setup is operated by a single external controller. The controller deploys forwarding rules to the switch to separate the racks' traffic and forward it according to the greedy routing. It also mimics the behavior of the distributed control plane.

6.4.1.2 ToR emulation concept

Generally, the emulation considers the two directions of each (10 Gbps) link as separate directed links, hence, ingress and egress directions can be assigned to different ToRs. For instance, in Figure 6.16, the egress direction of the port next to the ❸ is used by the green rack (small box); the port next to it is used by the orange rack. The assignment of the ingress direction is visualized by the color of the surrounding box, e.g., ToR1 (green) is assigned to the upper left group of ports.[9]

Static Packet Forwarding vs DA Forwarding Example. Rack 1 intends to send a packet to Rack 6; this forwarding goes along a 2-hop path (ToR1 → ToR3 → ToR6). First, it sends a packet on the NIC ❶. The Tofino receives the packet ❷ and forwards it internally on an outgoing port of ToR1 ❸. Please note here how the emulation exploits the differentiation of ingress and egress of the Tofino switch; the outgoing port of ToR1 is not located physically on the same QSFP port where the ingress happens. ToR4 receives the packet via the loopback cable ❹ and then sends it out to

[9]The decision to grouping ports (SFP+) by their QSFP port was done deliberately for readability. From an implementation point of view other groupings are possible too.

ToR6 ❺. After receiving the packet on ToR6 ❻, it is forwarded to Rack 6 ❼ and finally received ❽. Using *DA* links, the packet initially travels on the same path as before ❶❷ but ToR1 ❸ can directly transmit the packet to ToR6 ❹. This saves one hop. With the flexible P4-based ToR emulation, this setup can emulate static and *DA* de Bruijn DCNs.

6.4.2 Data & Control Plane Implementation

The P4 pipeline consists of two major parts, forwarding and elephant flow detection, which are described in the following.

6.4.2.1 Forwarding

A crucial aspect of emulating multiple ToRs on a single Tofino is to identify the location of the incoming packets in the emulated network, i.e., at which ToR they arrive. The prototype implements this by a first table lookup which exactly matches the ingress port of the packet and adds the ToR id to a metadata header. The actual de Bruijn forwarding uses a second table that combines the ToR id in the metadata header as an exact key along with an LPM on the packet's destination IP address.

The addressing used in the emulated network follows the approach that is presented in Section 6.2.3.2. Since the testbed can emulate a maximum of 16 ToRs, four bits are needed for the de Bruijn node ids. The 12 most significant bits are the base prefix, and the remaining 16 bits identify the hosts.

6.4.2.2 Elephant flow detection

The elephant flow detection algorithm always runs when a packet arrives at its destination ToR. If this packet belongs to an elephant flow, the data plane notifies the control plane using a digest, unless it is listed as an already reported flow.

Per-flow statistics. The prototype defines a flow as the aggregate traffic on the ToR-to-ToR level. Thus, for a small number of ToRs, it is feasible to keep track of the individual flows' statistics. The Tofino Native Architecture provides built-in functionalities for low pass filters that can be leveraged to estimate flow rates. Considering a maximum of 16 racks and ToR-to-ToR level flows, 256 individual low pass filters are needed to monitor every possible flow. For larger numbers of ToRs, probabilistic approaches such as sketches can be used.

Elephant flow threshold and detection. Using the per-flow statistics, the prototype classifies a flow as elephant if the number of packets sent in a given time interval exceeds a given threshold. To avoid the reverse direction of flows (the acknowledgements) and to reduce the efforts for parameter tuning, only packets larger than 255 B are included in the statistics. Note that both threshold and time interval are configurable and might benefit from optimizing for different traffic scenarios.

Preventing control plane flooding. The prototype features introduce a simple filtering mechanism to avoid flooding the control plane with digests. After an elephant flow is detected, it sets a `reported_elephants` register value from 0 to 1000 for this flow. As long as the threshold is exceeded, each subsequent packet from this flow decreases `reported_elephants` by 1. The data plane does not report the flow until this value reaches 0 again. Additionally, there is a `known_elephants` table with exact matching on the source and destination addresses. Upon detection, the control plane inserts a matching entry and thereby, mutes the digests for this flow. The initial value of the flow's `reported_elephants` register can be tuned to give enough time for the control plane to create the table entry.

6.4.2.3 Control plane implementation

The Duo prototype uses a Python-based centralized controller. The controller ignores the available global knowledge and implements the distributed *DA* link scheduling as presented in Algorithm 4 and 5 in a sequential way. That is, a single central processing unit (CPU) processes the elephant flow notifications from the switch. The controller maintains a list of the currently connected neighbors per-ToR. This list in turn is used to derive the forwarding tables with Algorithm 2.

When bootstrapping the network, the controller populates the tables for matching ingress traffic to the correct ToR and installs the forwarding entries following the greedy routing on a static de Bruijn topology. It then starts listening for elephant flow notifications.

Processing notifications. Upon reception of an elephant flow notification, the controller extracts the sending ToR's id t (destination of the elephant flow) and the flow's source ToR id s. In order to disable transmission of further notifications, it adds the flow to the `known_elephants` table. Afterward, it runs *DistDaRX* (Algorithm 4) with the state of the destination t to get the port request. The request is passed to *DistDaTX* (Algorithm 5) along with the state of the source s. If the request accepted, the controller obtains the *DA* link to create and mark the *DA* ports in s and t as reserved. Finally, the controller checks if any elephant flow has passed its cooldown period and removes it from the `known_elephants` table on the switch and resets the `reported_elephants` register to 1000.

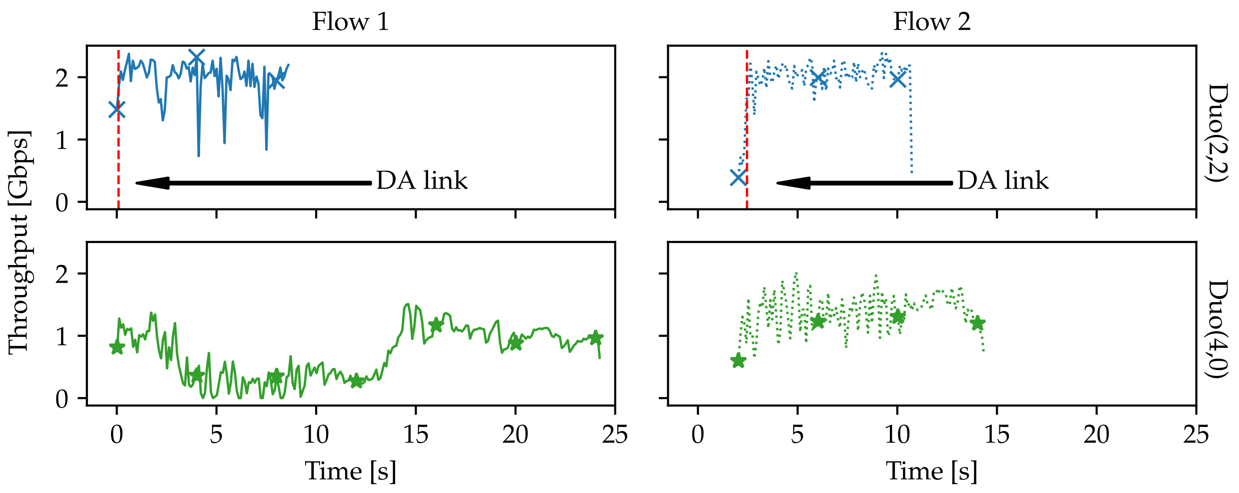
Setting *DA* links. The emulation implements *DA* links by (re-)assigning egress directions of ports to ToRs and, consequently, adding the needed forwarding rules (cf. Figure 6.16). Specifically, since the ingress ports of a ToR are fixed, the controller first infers the egress port that belongs to the loopback cable that connects to the ingress port of the *DA* link's destination ToR. Then, it updates the forwarding table of the *DA* link's starting ToR following Algorithm 2 and using this egress port (i.e., colors of *DA* ports in Figure 6.16 change over time). The current version does not emulate reconfiguration delay but can implement this by delaying the forwarding table updates. In case other *DA* links must be removed first, the forwarding tables of the involved ToRs are updated accordingly beforehand.

6.4.3 Measurement Results

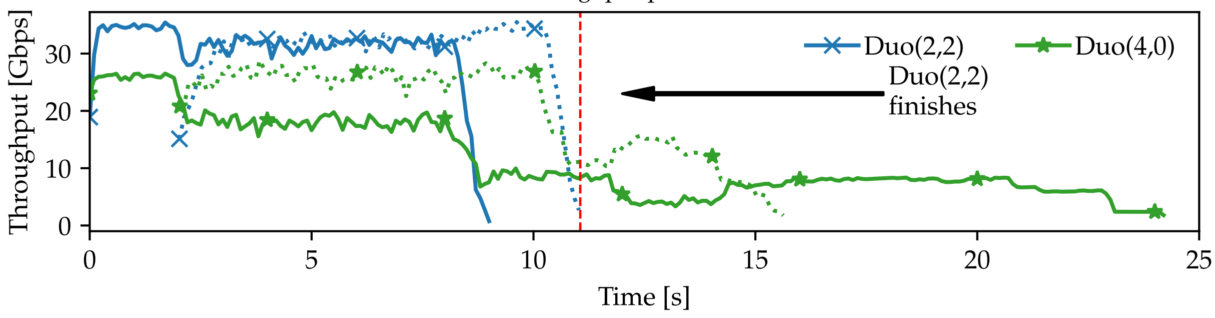
The following part validates the behavior of the proof-of-concept implementation and thereby, demonstrates the feasibility of Duo.

6.4.3.1 Settings and traffic input

The validation study considers a scenario with 16 ToRs and two configurations: Duo (4,0) and Duo (2,2). The rate on the ToRs' up-links, i.e., the ports that are connected via the loopback cables, is limited to 2.5 Gbps using traffic shapers available on the Tofino switch. The reconfiguration delay is $r_{da} = 0$ ms and *DA* links are reserved for $\rho = 200$ ms. Moreover, the threshold for elephant flows is 90 Mbps windowed over 0.5 s. This value turned out to be sufficient to separate the data



(a) Throughput per flow.



(b) Aggregate throughput per matrix.

Figure 6.17 Measurements of throughput over time. Results from measurements in a 16 ToR testbed with two bursts (matrices) of flows arriving. The colors compare Duo's configurations and linestyles distinguish the arriving matrices. The upper figures show two flows selected from the matrices, the lower figure shows the overall throughput. Transmission over Duo (2,2) finishes before Duo (4,0) since it can utilize the Da links.

streams from the flow of TCP acknowledgements. The cooldown time for elephant flows is set to 200 ms. During this period the data plane does not send elephant flow notifications to the control plane. The servers generate traffic according to two (different) permutation matrices which arrive with 2 s inter arrival time. Each matrix element corresponds to one TCP flow of 2 GB size. Note that the acknowledgements of the TCP flows can take different paths than the main data. The first matrix is constructed in such a way that all flows must use at least two hops on the static de Bruijn topology. The second matrix is randomly generated.

6.4.3.2 Analysis

Figure 6.17 compares the achieved throughput for the two configurations of Duo. Figure 6.17a focuses on the throughput per flow over time. For each configuration, it shows the two flows from both matrices which originate at Rack 2. The flow of the first matrix is shown in the left column, the one of the second matrix in the right column. The upper row is Duo (2,2), the lower one Duo (4,0).

For Duo $(4, 0)$, we observe that the two flows share at least one link, after the second flow has arrived in $t = 2$s. Neither of the two flows reaches the full link rate and, hence, both have a transmission duration of 24.22 s and 14.33 s respectively – significantly longer than the theoretic optimum of 6.4 s. In particular, the first flow suffers from interference with other flows. Even after the second flows has finished, its rate remains around 1 Gbps which hints at additional congestion on another link. In contrast, Duo $(2, 2)$ uses the elephant detection to identify large flows and establishes *DA* links for each flow. In the figure, the vertical, red dashed line indicates when the *DA* link for the respective flow is set up. As a result, both flows send at link rate and, hence, finish earlier at 8.6 s and 10.73 s. This demonstrates that Duo reacts appropriately to the elephant flows it detects.

Taking a broader perspective, Figure 6.17b compares the aggregated throughput per permutation matrix (here a set of 16 flows). The dotted and solid lines discriminate the matrices (solid: first matrix, dotted: second matrix), whereas markers and colors indicate Duo's configuration. Compared to Duo $(4, 0)$, Duo $(2, 2)$ achieves higher throughput of $\approx$ 30 Gbps per matrix by setting *DA* links. The vertical, red dashed line indicates when the transmission with Duo $(2, 2)$ finishes. The overall demand completes $\approx$ 54% earlier than for Duo $(4, 0)$. For Duo $(4, 0)$, the throughput for the first matrix (green, solid) decreases when the second matrix arrives. Overall this demonstrates that Duo is realizable and underlines the benefits of *DA* reconfigurations.

6.5 Discussion of shortcomings and open questions

This section discusses several important open questions that this study leaves for future research to improve Duo.

Classification of flows. Duo assumes that an a priori classification into latency sensitive and throughput sensitive flows is available, based on which the transport protocol and queue priority is chosen (similar to Opera [49]). This classification can be done using application-level information but for the sake of this thesis, the flow size is considered. Obtaining the size of a flow in advance is still considered a challenging problem in the community and several methods such as aging or machine learning have been proposed [312]. In general, a mis-estimation of the flow size can impact the performance of Duo. Namely, sending large flows via the high priority queues will block latency sensitive traffic, as also illustrated by the sensitivity analysis of f_{th}.

Scalability of existing networks. Since Duo builds around a de Bruijn graph, the ability to scale-out the network, i.e., add nodes, is somewhat limited. First, all spines switches are connected to all ToRs, i.e., their size must potentially be increased. Second, recall that a b-ary de Bruijn graph of dimension d has b^d nodes. Adding nodes can either be achieved by adding static switches (increase b) or increasing the dimension d. In both cases, the forwarding tables and potentially also the static matchings have to be updated.

Failures. Duo is in general expected to have good connectivity in case of failures. Considering link failures, de Bruijn networks are known to provide good robustness properties. Moreover, a failed static link can be replaced by a *DA*-link to maintain connectivity in the static de Bruijn topology (failed *DA*-link can simply not be used anymore). For switch failures, a static switch can be fully replaced by a dynamic one to maintain the connectivity. Nevertheless, while connectivity can be maintained using the dynamic links and switches, some capacity will be lost during failure. We leave a detailed empirical study for future work.

6.6 Summary

Many of the existing proposals for RDCNs come with limitations and overheads. To address these, this chapter presents Duo, a high-throughput topology that features a flexible architecture with static and *DA* links which supports integrated multi-hop routing. Duo builds around the structural properties of de Bruijn topologies that essentially allow performing updates of the nodes' forwarding tables with local information only.

This chapter presents both a centralized and a distributed algorithm for the scheduling of *DA* links and provides details on a possible implementation. The simulation-based evaluation demonstrates the benefits of Duo under various traffic conditions and illustrates that the performance gains indeed stem from the use of integrated multi-hop routing. As shown by the simulations and the prototype implementation, Duo does not require complex buffer management schemes as related state-of-the-art designs but can be implemented using available COTS equipment.

Duo can be seen as a next step towards more practical *DA* RDCNs. It opens several opportunities for future work. First, presented design and evaluation focus on the performance of Duo but leave aside resilience to failures. Moreover, while the performance with standard TCP was explored thoroughly the behavior and possible, further performance improvements with more specialized transport protocols may be an interesting aspect to explore. Also, it might be interesting to explore other network topologies that support greedy routing.

Finally, although the evaluation covers already some traffic patterns, it neglects the special requirements of applications exhibiting uniform communication patterns as they are explored in the course of the next chapter.

Chapter 7

Designing Topology Reconfiguration-aware Networks for Datacenters

This chapter addresses the third characteristic of reconfigurable networks with programmable topologies, the class of reconfiguration. As concluded in Chapter 3, there are two major classes of topological reconfiguration in prior work, demand-aware (*DA*) and demand-oblivious (*DO*) reconfigurations. Whereas there are only *DA* solutions for the wide area network (WAN) use case, there are examples of both classes for reconfigurable datacenter networks (RDCNs). Hence, this chapter focuses on the datacenter network (DCN) use case.

The traffic in DCNs is not only growing explosively as an aggregate but also embodies complex traffic patterns that come in many flavors with different performance requirements [37, 35, 34]. We have already investigated one example of such traffic in Chapter 2. The distributed machine learning (DML) applications presented there use the all-reduce communication primitive that results in ring- or tree-like traffic patterns. Other applications, such as *Hadoop*, exhibit all-to-all traffic patterns [34], and short query traffic has been shown to have a skewed pattern [311]. High-performance proposals for new DCN architectures must simultaneously satisfy all the different requirements (e.g., low latency or high throughput) from these traffic patterns.

At the same time, there exist several fundamentally different optical datacenter (DC) topologies using different switching technologies (cf. Chapter 3 and [119]). Besides the classification in *static* and *dynamic*, i.e., reconfigurable topologies, we can distinguish them by their reconfiguration-class (*DO* and *DA*). Using these two dimensions, the large number of examples of RDCNs in prior work has condensed into three major groups: (1) *static* and *DO* topologies resemble the traditional architectures like clos and expander graphs [211, 42, 313, 314, 44]. The second group is given by *dynamic* and *DO* topologies such as [48, 49, 53, 173] (cf. Chapter 3). Finally, there are *dynamic* and *DA* topologies [176, 315, 177, 46, 47, 175].

However, it must still be explored how these designs compare for specific traffic types. In particular, this chapter is motivated by the observation that a topology built from a representative of a single group is not suited for all traffic patterns. For example, small (mice) flows (e.g., from short queries) are latency-sensitive and should not be transmitted on dynamic topologies to avoid waiting times due to reconfigurations. Hence, they should use a static topology. Conversely, large (elephant) flows benefit from short paths or even direct connections, as provided by dynamic *DA* topologies. Here, the reconfiguration time is small compared to the (ideal) flow transmission time and can be amortized by higher throughput. Lastly, it has been shown that dynamic and *DO* topologies with periodic direct connectivity between *all* rack pairs are particularly well suited for traffic with all-to-all patterns [48, 49, 53]. In summary, the best achievable network performance depends on the topology and the present traffic. Moreover, the current state-of-the-art lacks RDCN designs that introduce reconfiguration-awareness, at least on a macroscopic level, i.e., *DA* vs. *DO*.

Motivated by the identified inefficiencies resulting from a mismatch between traffic and topology, this chapter presents a hybrid RDCN architecture containing multiple classes of reconfigurable (sub-)topologies suited for different traffic types.

Specifically, the novel architecture Cerberus[1] facilitates serving traffic on the topology part that best *matches* the traffic's characteristics. For example, latency-sensitive mice flows can be transmitted via static switches, all-to-all traffic via dynamic *DO* switches, and elephant flows via *DA* switches. An initial evaluation using flow-level simulation shows that Cerberus outperforms alternative architectures by achieving higher throughput [316].

Afterward, this chapter explores how to implement such a hybrid RDCN in more detail. Therefore, it presents Trio, an end-to-end system design for Cerberus, including a data plane architecture and integrating Duo for the static and *DA* topology part. Moreover, the final design contains an alternative scheduling algorithm for the *DO* topology part that reduces the communication overhead and comes with network and transport layer designs to facilitate the adaptation of the sub-topologies toward evolving traffic. Trio is evaluated using packet-level simulation to demonstrate the benefits of matching traffic and topology, even over time, rendering RDCNs reconfiguration-aware on a macroscopic level. Specifically, this chapter shows that Trio achieves higher throughput than state-of-the-art topologies as well as competitive flow completion times.

Content and Outline The content of this chapter is based in parts on one previous journal publication [3] and one other publication currently under submission [4]. The first work [3] is also part of another dissertation. Hence, only parts of it are presented here. While the motivation and the design are part of both theses (with equal contribution of both authors), this thesis contributes the simulative analysis. The parameter optimization and the derivation of the analytical performance bounds are part of the other thesis. From the second work [4], this thesis presents the design and packet-level simulative evaluation. Again, the analytical evaluation is part of of the other thesis.

Specifically, Section 7.1 presents a more detailed motivation for this chapter re-using our previous analysis from [3, Section 3]. Section 7.2 gives an overview of related work and is an addition of this thesis. Section 7.3 presents the Cerberus concept. Therefore, it first introduces the topology components and the needed algorithm and, finally, shows initial evaluations. This section re-uses contents from [3, Section 4 and 6]. Finally, Section 7.4 takes Cerberus' idea further and presents and evaluates an integrated end-to-end design called Trio. This last section builds on [4, Sections 3, 4, A and B], which is currently under review. Finally, Section 7.5 summarizes the chapter.

The implementation of Cerberus has been made publicly available at `https://github.com/tum-lkn/cerberus`.

[1]In Greek mythology, Cerberus is a dog with three heads (corresponding to our three topology parts).

7.1 Motivation

The main motivation for introducing macroscopic reconfiguration-awareness, and with this for Cerberus, is the observation that specific topologies are better suited for some traffic classes than others. To investigate this more in detail, we consider the end-to-end throughput in a fluid-level model as the main performance metric since it is of high interest to DCN operators [316, 43] and also hints at a topology's resource efficiency and flexibility [43]. This section recapitulates the analytical expressions (upper bounds) for the throughput of three representative DCNs (and thereby, the different classes of reconfigurations) that have been derived in our previous work [3, Section 2] but are part of a different thesis. The expressions hint at how these topologies perform for different traffic patterns. There are two representatives for *DO* topologies: expander-based topologies (*expander-net*) like [211, 314] for static and RotorNet [48] (*rotor-net*) for dynamic topologies. A fully reconfigurable *DA* topology built only from k *DA* switches serves as the representative for *DA* RDCNs (*demand-aware-net*) like [188, 47]. It allows the creation of any k-regular directed graph. The following recapitulates the main findings from [3] about their throughput.

The analysis uses the throughput definition for fluid-flow systems from [316]. It defines the throughput for a demand matrix T $\theta(T)$ as the largest scaling factor such that there is a feasible multi-commodity flow assignment that obeys link capacities and flow conservation. [316].[2] Considering a DCN with n racks and the respective amount of top-of-racks (ToRs), the demand can be represented by an $n \times n$ demand matrix $T = [t_{uv}]$ whose entries are given as rates in bps. Each ToR has k *links* of rate R connected to other switches and k links connected to servers in its rack. Further, consider the *hose* model [316] and assume for now that T is *saturated*, i.e., the sum of rates in each row and column of the demand matrix is kR. $\hat{T}$ is the demand matrix in *bits* for *one second*, Total($\hat{T}$) denotes the total number of bits in $\hat{T}$, and DCT($\hat{T}$) is the total demand completion time (DCT) of $\hat{T}$ with a feasible multi-commodity network flow.

In this scenario, the following two relations between DCT and throughput have been proven [3]: i) If DCT($\hat{T}$) $\geq x$ for every feasible DCT of $\hat{T}$, then: $\theta(T) \leq \frac{1}{x}$, ii) if there exist a feasible network flow s.t. DCT($\hat{T}$) $\leq y$, then $\theta(T) \geq \frac{1}{y}$.

These results help to obtain the following theorems about the throughput of *expander-net*, *rotor-net*, and *demand-aware-net*, respectively:

- For an *expander-net* formally defined as $G(k)$, a (random) k-regular expander with n nodes, the throughput is upper bounded by $\theta^* \leq \frac{1}{\text{epl}(G(k))}$, where epl(.) is the *expected (average) path length* of $G(k)$.

- For a *rotor-net* with reconfiguration delay r_{do} and slot time s, the throughput is bounded by $\theta^* \leq \frac{n}{2n-1}\frac{s}{r_{do}+s}$. For a given demand matrix T, $\theta(T) \leq \frac{1}{2-\phi(\hat{T})}\frac{s}{r_{do}+s}$, where $\frac{1}{n} \leq \phi(\hat{T}) \leq 1$ is the *traffic skewness* of T. Informally, ϕ denotes the fraction of bytes in *rotor-net* which is sent through a single hop and $1 - \phi$ is the fraction which is sent via two hops using Valiant routing [48].

- For a *demand-aware-net* with reconfiguration delay r_{da} of the individual links that can be reconfigured independently if both source and destination have an available port, the

[2]Note that this definition does not capture propagation or queuing delay of individual flows as it targets on a macroscopic performance value for the topology. These delays will be captured by the empirical evaluations in Section 7.4.2.

throughput can be bounded by $\theta(T) \geq \frac{k}{\sum_{j=1}^{m}(r_{da}+\frac{w_j}{R})}$. Here it is assumed that the demand matrix T is a union of m permutation matrices T^j each of which can have a different weight w_j. Formally, $T = \sum T^j$.

Starting with *expander-net*, we observe that the upper bound is inversely proportional to epl. Hence, the throughput reduces due to multi-hop routing, the so-called *bandwidth tax*. For *rotor-net*, the throughput is reduced by $\phi(\hat{T})$ and r_{do}. The latter indicates a dependence on the latency tax. As elaborated in [3], also $\phi(\hat{T})$ can serve as a proxy for the bandwidth tax as it indicates how much traffic is sent via 1- or 2-hop paths. It equals 1 for the all-to-all demand matrix (uniform demand), which results in the maximum throughput. Thus, the throughput bound strongly depends on the demand T ranging $\theta^* \leq \theta(T) \leq \frac{s}{r_{do}+s}$ where the worst matrix is the permutation matrix and the best matrix is the uniform matrix.

For *demand-aware-net*, the reconfiguration time (latency tax) is critical for the throughput. Moreover, the throughput can change significantly as a function of T. In contrast to *rotor-net*, the best case for *demand-aware-net* is when T is an arbitrary, single permutation matrix. In this case, only one reconfiguration is needed, and hence, the throughput is close to one ($\frac{1}{r_{da}+1}$). A static network could potentially deal with a single permutation matrix T optimally if it is *known a priori*. *Demand-aware-net* will deal with any such T optimally, even when it is *unknown* a priori. The worst case for *demand-aware-net* is when T is the uniform matrix and built from a collection of n permutations matrices of low weight, i.e., each of weight $\frac{kR}{n}$. Plugging this value of w_j into the expression will give us a lower bound of $\frac{k}{nr_{da}+k}$, and the throughput can be close to zero. Overall, the throughput is bounded by $\frac{k}{nr_{da}+k} \leq \theta(T) \leq \frac{1}{r_{da}+1}$.

The main observation from the previous recapitulation is that different existing DC topologies can have different advantages and disadvantages depending on the scenario. This motivates the presented novel RDCN design that is tailored toward the specific setting and traffic it serves. The design mainly builds around the hypothesis that throughput in DCNs can be significantly improved if the *network topology matches the demand*. The following discussion elaborates on the potential inefficiencies arising from a mismatch of traffic patterns and topologies. In particular, using an empirical example provides more intuition on how "taxes" can be used to quantify inefficiencies.

Figure 7.1 visualizes the different flow size distributions for WEBSEARCH [311], DATAMINING [193], and HADOOP [34] applications. Flow sizes and their distribution can vary widely across and within applications. For instance, in the case of DATAMINING, about 75% of the traffic belongs to flows of sizes above 100 MB, whereas all flows are smaller than 40 MB in WEBSEARCH. Considering an ideal situation (e.g., a dedicated link per flow), the flow size translates directly into the flow transmission time. The top abscissa (x-axis) in Figure 7.1 shows the ideal flow transmission time on a 40 Gbps link. We observe that it can range from microseconds for small flows up to seconds for elephant flows.

Influence factors for the flow transmission time (and hence, the throughput) are not only the flow size and the links speed but also the optical switching technology used. In particular, the reconfiguration delay present in dynamic topologies has a major impact. While the exact values depend on the specific technology, the reconfiguration delays of *DA* topologies (like [45, 191]) are likely higher than those of *DO* topologies (like [53, 48, 49]).[3] Whether a flow can profit from reconfigurations depends on the ratio between its size (i.e., transmission time) and the latency

[3]See also the elaborations on related work in Chapter 4.

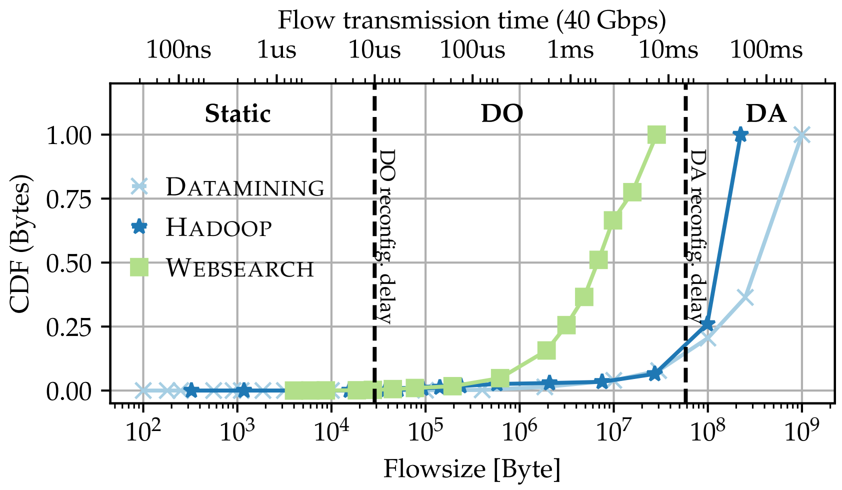


Figure 7.1 Cumulative distribution function (CDF) of the number of bytes transmitted according to various flow size distributions. The top x-axis shows the ideal flow transmission time on a 40 Gbps link. The figure is partitioned into three parts (Static, *DO*, *DA*) based on the reconfiguration delay of *DO* (rotor) and *DA* switches. The partition indicates the "best" topology (reconfiguration class) for flows of the the respective size.

tax. For example, an elephant flow has a large transmission time compared to the reconfiguration delay. Hence, the reconfiguration can be amortized and may pay off in the long run.

As summarized above, static topologies do not introduce any latency tax but a bandwidth tax [48]: since flows potentially must traverse multiple hops to reach the destination, additional network capacity is consumed. Dynamic, *DO* topologies can reduce the bandwidth tax by providing temporary single-hop paths between node pairs and avoiding multi-hop forwarding. Such approaches perform particularly well for uniform all-to-all traffic patterns [48, 49]. However, they are not optimal for elephant flows, which usually make the majority of the traffic volume rendering their optimization critical (cf. Datamining in Figure 7.1). Indirect routing approaches like Valiant load balancing (VLB) [171] can improve the performannce of *DO* RDCN for elephant flows but again introduce bandwidth tax (cf. the upper bound for *rotor-net*). In contrast, *DA* reconfigurations facilitate setting up direct shortcuts that serve the elephant flows. To conclude, *DO* topologies are better suited for uniform demands, whereas *DA* reconfigurations should be preferred for skewed demand with large flows that can amortize latency taxes

Figure 7.1 provides an example of the observations above. The dashed vertical lines show the reconfiguration delays of (rotor) switches (10 µs) and *DA* switches (15 ms) and partition the figure in three areas, which indicate the best topology for the respective flow sizes. Elephant flows are in the right part of the figure (*DA*) starting at 15 ms. The flows in this area are can amortize this latency tax, e.g., the tax for a flow of 500 MB whose ideal transmission takes 100 ms is 15%. Creating a direct link for such flows will reduce the bandwidth tax to a minimum. The middle part (*DO*) of the figure belongs medium-sized flows that benefit from *DO* reconfigurations. If we consider a reconfiguration delay of 10 µs and a slot time of 100 µs (91% utilization) a cycle through all possible links is still fast and in the order of the flow transmission times that range between 10 µs to 15 ms. In addition, the medium-sized flows are the vast majority of flows in the probability distribution. If the source-destination pairs are sampled uniformly, the resulting traffic resembles an all-to-all pattern which is particularly suited for the rotor switch's connectivity

Table 7.1 Overview of related (R)DCN designs and their properties.

Name	Static	*DO*	*DA*	Reconfiguration delay	Granularity
Xpandr [211]	✓			-	-
Jellyfish [314]	✓			-	-
Hypercubic [313, 317]	✓			-	-
RotorNet [48]	✓	✓	✗	µs	-
Opera [49]	(✓)	✓	✗	µs	-
Sirius [53]	✗	✓	✗	ns	-
Helios [45]	✓	✗	✓	ms	Matrix (Matching)
OSA [46]	(✓)	✗	✓	ms	Matrix (Matching)
Mordia [52]	✓	✗	✓	µs	Matrix (Schedule)
Eclipse [188]	✓	✗	✓	n.d.	Matrix (Schedule)
ProjecToR [47]	(✓)	✗	✓	µs	Flow
C-Through [56]	✓	✗	✓	ms	Matrix (Matching)
MegaSwitch [54]	✓	✗	✓	ms	Matrix (Matching)
xWeaver [191]	✓	✗	✓	ms	Matrix (Matching)
ReNet [318]	✗	✗	✓	n.d.	Flow

pattern. Finally, the left area (Static) shows small flows. These flows would suffer if exposed to reconfiguration, but since their cumulative volume is small, accumulating bandwidth tax from multi-hop forwarding is not critical for the overall resource efficiency. For example, they can be forwarded via a static expander topology with a short average path length, therefore, enabling low-latency forwarding. The exact flow size thresholds for making forwarding decisions can be computed analytically as summarized in Section 7.3.3.

7.2 Related Work

Section 3.5 provides a broad overview of prior work in the realm of RDCNs. In order to discuss more in detail how Cerberus and Trio differentiate from these, this section focuses on the included reconfiguration classes and control plane complexities.

Table 7.1 summarizes the characteristics of major prior work. It lists the used reconfiguration classes, the achieved reconfiguration periods, and granularity. As a reference, the first part shows examples of static topologies which are widely deployed, such as Clos topologies and multi-rooted fat-trees. Such designs have recently also been complemented by modular hypercubic networks [313, 317] as well as expander-based solutions [314, 211]. These topologies use a single topology type without reconfigurations and typically rely on multi-hop routing to provide connectivity between all nodes.

The second group revolves around dynamic, *DO* topologies. RotorNet [48] is a first example that provides high bandwidth by emulating a full-mesh network over time. Its *DO* nature restricts the reconfiguration granularity to ToR-to-ToR level but allows reconfiguration periods in the order of microseconds. The follow-up work, Opera [49], improves the idea of RotorNet. It features a deterministic reconfiguration scheme that ensures connectivity between the nodes at any time. Therefore, it implements an expander graph that changes over time to temporarily provide single-hop paths between all ToRs. Effectively, this results in a hybrid design in which the *DO* topology part is complemented by a static network to serve low-latency traffic. Sirius [53] employs a similar idea of a *DO* topology as RotorNet. However, Sirius' implementation differs as it reconfigures by adjusting the sending wavelength on the transmitting node. The passive core built from arrayed waveguide grating (AWG) routers guides the light to the respective receiving node. This approach enables reconfigurations in the order of nanoseconds.

Considering *DA* RDCN, we first observe that there are differences in the granularity of reconfigurations. Two of the listed solutions, Eclipse [188] and Mordia [52], are more coarse-granular. They rely on a (predicted) traffic matrix to schedule a sequence of *DA* configurations. In contrast, more fine-granular designs such as OSA [46], C-Through [56], xWeaver [191], MegaSwitch [54], or Helios [45] determine a single configuration for the predicted matrix. Finally, ReNet [318] and ProjecToR [47] are the most fine-granular solutions. They set up *DA* links on a per-flow basis, also in combination with decentralized reconfigurations [47].

Due to the increased reconfiguration time experienced in *DA* RDCNs, many of these solutions additionally rely on a static topology part for control plane and or low-latency traffic. This part might either be realized explicitly by means of additional electronic packet switches (EPSs) or implicitly by fixing the configuration of specific links (or changing them more slowly). Examples of the former approach are Helios [45], C-Through [56], xWeaver [191], and Flexspander [175]. The type of the static topology may vary, e.g., expanders or Clos topologies. Examples of solutions that provide the static topology part implicitly are ProjecToR [47], OSA [46], and MegaSwitch [54]. The former always maintains a "base mesh" of connected links that can handle low-latency traffic. In contrast, OSA and MegaSwitch allow reserving some circuit-switch ports specifically to ensure connectivity for low-latency traffic.

In contrast to the previous work, this chapter combines and integrates three types of topologies and thereby, multiple reconfiguration classes and studies the consequences of mismatching traffic to specific network types using a unified model.

7.3 Cerberus: The Power of Choices

This section describes and evaluates Cerberus a RDCN design that combines multiple reconfiguration classes. First, this section introduces the topology components more in detail (Section 7.3.1) and presents a flow assignment algorithm (Section 7.3.2). It then summarizes how to obtain a meaningful parametrization (Section 7.3.3) and finally, evaluates Cerberus using flow-level simulations (Section 7.3.4).

7.3.1 Cerberus' Topology Components

Cerberus uses the ToR-Matching-ToR (TMT) model described in Section 3.6. It combines three sub-topologies belonging to different reconfiguration classes: a static part, a *DO* part, and a *DA* part. These sub-topologies may either be implemented by different switch technologies (e.g., rotor switches for *DO* and full crossbar optical circuit switches (OCSs) for *DA*), or by a single switch technology that supports multiple *modes of operation*. The former may be more cost-effective (e.g., static topologies are cheaper), while the latter is more flexible.

Following the TMT, we can describe the three sub-topologies by a collection of spine switches. Each spine switch type is defined by a 4-tuple $sw = (m, \mathcal{M}, s, r)$ where $\mathcal{M}$ is the specific set or sequence of the m matchings the switch can support; s is the minimal circuit-hold time a switch needs to remain in a matching that contains a specific link before switching to the next matching which does not contain this link; r is the reconfiguration delay. The three sub-topologies can be formalized as follows:

- **Demand-aware topology (Da):** The *DA* sub-topology consists of a collection of k_{da} Da reconfigurable switches that are all described by the tuple $sw = (n!, \mathcal{M}, S, r_{da})$. *DA* switches can reconfigure to *any* of the $m = n!$ possible directed (perfect) matchings. The *DA* switch can be implemented using commercially off-the-shelf (COTS) 3D MEMS technology with reconfiguration delay in the order of tens of milliseconds. This section assumes $r_{da} = 15$ ms which is the typical reconfiguration delay of a 3D MEMS switch [222, 122]. The circuit-hold time s can change during the operation of the *DA* switch. While in this case the minimal time is zero, as a rule of thumb $s \gg r_{da}$ for the reconfiguration to be worthwhile. Otherwise, the utilization of the link would be small as most of the time is spent on reconfiguration.
- **Demand-oblivious (rotor) topology (Dobl):** The *DO* sub-topology consists of the union of k_{do} rotor switches, henceforth called Dobl switches. Each switch is described by the tuple $sw = (n-1, \mathcal{M}, s, r_{do})$: a Dobl switch cycles through $n-1$ matchings specified by $\mathcal{M}$, emulating a fully-connected network (i.e., complete graph) and, hence, providing high bandwidth to all-to-all traffic. The Dobl switch is a slight generalization of the original rotor switches [48]. The present model uses $n-1$ matchings and not $\frac{n}{k}$ matchings as proposed originally.[4] The reconfiguration delay for the Dobl switch is in the order of microseconds. This section assumes $r_{do} = 10\,\mu s$ as in [49, 48]. The slot time of an Dobl is tunable. A reasonable setup is at least $s = 9r_{do} = 90\,\mu s$ to reach 90% amortization of the reconfiguration delay [49, 48]. Note that the bounds recapitulated in Section 7.1 hold for any setting of r_{do} and s.
- **Static topology (Static):** The static topology is a union of k_s static matchings, where each matching can be implemented, e.g., using an optical patch panel or even static electrical connections. A static switch can be represented by: $sw = (1, \mathcal{M}, \infty, 0)$. $\mathcal{M} = \{M\}$, i.e., it has a single ($m = 1$) predefined matching that does not change over time ($s = \infty, r = 0$). The static switches are cost-effective components to create regular graphs, such as expander graphs, providing low latency for short flows using multi-hop routing. Prior work has shown that good expanders can be obtained by taking the union of a few matchings [319].

Let (k_s, k_{do}, k_{da}) denote a TMT network consisting of k_s Static switches, k_{do} Dobl switches, and k_{da} Da switches. This section will refer to a network with only Static switches, i.e., $(k, 0, 0)$, as *static-net*. The static topology component of Cerberus specifically relies on expander graphs, and hence, this network will be referred to as *expander-net*. Further, a network consisting of only Dobl switches, $(0, k, 0)$, is referred to as *rotor-net*, and a *demand-aware-net* is a topology consisting only of Da switches, i.e., $(0, 0, k)$. On the contrary, Cerberus uses a mix of switch types, (k_s, k_{do}, k_{da}), where $k = k_s + k_{do} + k_{da}$.

With these specific sub-topologies, the TMT network can model many existing systems. Examples of *rotor-net* are RotorNet [48], Opera [49] and Sirius [53], which rely on pre-defined periodic sequences of matchings. Networks like ProjecToR [47], Eclipse [188], or Helios [45] can be modelled as *demand-aware-net*. ProjecToR additionally uses a static electric network, which in the TMT model can also be described using an *expander-net*. The TMT also applies to static topologies like Xpander [211], which can be modelled as an *expander-net* (even though it is based on electrical switches).

[4]We explain why this generalization improves the performance of *RotorNet* in [3, Appendix B]

Algorithm 6 Cerberus flow assignment

```
1:  Switch depending on flow size
2:    Case small flow:                                   {latency-sensitive flow}
3:      send to Static                                               {multi hop}
4:    Case medium flow:
5:      send to Dobl topology                                {using 1 or 2 hops}
6:    Case large flow:
7:      If a direct link is available to reconfigure:
8:        send to Da topology                                       {single hop}
9:      Else                                    {Under-provisioned demand-aware}
10:       send to Dobl topology                                {using 1 or 2 hops}
```

7.3.2 Cerberus' Flow Assignment Algorithm

Given a (k_s, k_{do}, k_{da}) network, this part describes the high-level flow assignment algorithm. Cerberus assigns flows to the three topology parts based on three categories: *small* (s), *medium* (m) and *large* (ℓ) flows. The flow size *thresholds* to discriminate flows into these categories are denoted by t_m and t_ℓ. Namely, *small* flows are of a size less than t_m, *medium* flows are of size more than t_m and less than t_ℓ, and *large* flows are of size larger than t_ℓ. The next section summarizes how to obtain the exact values for these thresholds.

Algorithm 6 describes how Cerberus distributes the traffic classes among the three switch types: small, latency-sensitive flows are forwarded via a static expander built from k_s Static switches; large flows are transmitted via the k_{da} Da switches in the system; and the remaining (medium) flows describing, e.g., all-to-all traffic which is not latency-sensitive, are routed via the k_{do} Dobl switches. Cerberus manages the large flows using an approach that can be seen as a distributed link cache: when a new *DA* link needs to be established, an existing link must be replaced or "evicted." While this introduces interesting optimization opportunities, this section focuses on a simple strategy. When a large flow should be sent to the Da switches, but there are no available ports to serve it (the related source/destination ports already serve other flows), Cerberus greedily transmits the large flow via the Dobl switches. When Cerberus forwards large flows continuously via the Dobl switches, the Da switches are *under-provisioned*. The next section describes how to obtain the optimal number of Da switches k^*_{da} to match the topology to the demand and, thereby, maximize the throughput.

7.3.3 Throughput Analysis and Parametrization

This section summarizes how Cerberus' logic calculates the size of each sub-topology (Static, Dobl, and Da) and the flow assignment thresholds for each topology.[5] It also lists expressions for the throughput and DCT of the system as a whole. To do this, we first describes the general traffic generation model that the analytical results hold for.

[5]Note that these contributions are part of another thesis but are summarized here to provide a description of the whole system to the reader.

7.3.3.1 Traffic Generation Model and Metrics

In the considered traffic generation model, flows arrive over time according to a sequence $\omega = (f_1, f_2, \ldots)$. The f_i are the individual flows which are described by their source ($s_i \in N$) and destination rack ($d_i \in N$), their volume v_i (bytes), and their arrival time t_i: $f_i = (t_i, s_i, d_i, v_i)$. Note that although servers originally generate the flows, the model employs a rack-level perspective. More formally, let $\mathcal{T}(x, L, \mathcal{D})$ be a traffic generation model where $x \in [0, 1]$ is the fraction of active ToRs, $L \in [0, 1]$ is the load of each active ToR, and $\mathcal{D}$ is a flow size distribution.

Specifically, $\mathcal{T}$ generates traffic by first selecting a subset of $x \cdot N$ ToRs as active, where x is the fraction of active ToRs. Then, for every active ToR, it generates traffic at an average rate of $L \cdot k \cdot R$. The source-destination pairs are sampled uniformly at random out of the set of active ToRs. Flow sizes are independently sampled from the considered flow size distribution $\mathcal{D}$. The flow arrivals follow a Poisson process, i.e., the inter-arrival times are sampled from a negative exponential distribution. The average arrival rate depends on the load L, the link rate R and the flow sizes. Accordingly, this model generates a total traffic of $(xN) \cdot (L \cdot k \cdot R)$ bits per second.

As stated before, CERBERUS maximizes the end-to-end *throughput* in a fluid model. The used definition is similar to the one from [316] but has been extended from a demand matrix to the general traffic generation model above and dynamic network topologies in [3]. Note that this definition focuses on the network topology and ignores packet-level effects such as ramp-up times from congestion control.

For the traffic generation model $\mathcal{T}(x, L, \mathcal{D})$ and a given fraction of active racks x, the throughput of a system is the maximum load L that the system can still serve. That is, the largest L for which a feasible solution of the multi-commodity flow problem exists that routes the flows generated by $\mathcal{T}$ subject to flow conservation and link capacities. Using Theorem 1 in [3], this condition can be reformulated in dependence on the DCT: The system has throughput of at least L for x, if the DCT of the demand matrix $\hat{T}$ that is built from the flows arriving in one second, is DCT($\hat{T}$) $<\leq 1$ [3].

7.3.3.2 CERBERUS' Throughput and Optimal Parametrization

Using the observation in the previous section, expressions for the throughput and the parametrization of the topology can be obtained. This part focuses on the case of $x = 1$. This section only briefly recapitulates the main expressions as the more detailed derivation and proof are part of a different thesis [3]. There, the author also shows the throughput proportionality of CERBERUS.

In order to understand the expressions, the following notation is introduced: $\hat{T}(L)$ is the accumulated demand matrix built from the flows generated by $\mathcal{T}(1, L, \mathcal{D})$ in one second. The DCT of a system $sys \in \{\textit{expander-net}, \textit{rotor-net}, \text{CERBERUS}\}$ is denoted as DCT($sys, \hat{T}(L), k$), where we recall that the systems can be specified in terms of the TMT model as *expander-net* $= (k, 0, 0)$, *rotor-net* $= (0, k, 0)$ and CERBERUS $= (k_s, k_{do}, k_{da})$. The symbol $\tau \in \{s, m, \ell\}$ denotes the category of a flow and $\hat{T}(L, \tau)$ is the expected number of bytes from flows in $\hat{T}(L)$ belonging to category τ. $\phi(\hat{T})$ is the expected *traffic skewness* of $\hat{T}$ resulting from the flow size distribution $\mathcal{D}$. ϕ can be interpreted as the fraction of packets (or bytes) subject to single hop transmission in *rotor-net*, and $1 - \phi$ is the fraction of packets (or bytes) that are sent over two hops using VLB [48]. The skewness can be approximated using the variation distance from the uniform distribution. More details are provided in [3].

Having this additional notation, the following expressions have been derived [3]: The expected throughput of Cerberus for a given T is

$$\theta(T) = \frac{\hat{T}(1,\ell)}{nk^*_{da}}\left(r_{da}\mathbb{E}\left[\frac{1}{|f|}\right] + \frac{1}{R}\right) \tag{7.1}$$

where the expectation of the reciprocal flow sizes considers only large flows and k_{da^*} is the optimal number of Da switches. In order to obtain the throughput, the threshold for the size of medium flows is

$$t_m = sR \tag{7.2}$$

which is the number of byte that can be transmitted in a flow on the Dobl switches. The threshold for large flows is

$$t_\ell \geq \frac{r_{da}\cdot t_m \cdot R}{(2-\phi)\cdot R\cdot(r_{do}+s) - t_m} \tag{7.3}$$

The ratio between the optimal number of Da switches, k^*_{da}, to the optimal number of Dobl switches, k^*_{do} is given by:

$$\frac{k^*_{da}}{k^*_{do}} = \frac{\hat{T}(1,\ell)}{\hat{T}(1,m)/t_m}\cdot\frac{r_{da}\mathbb{E}\left[\frac{1}{|f|}\right]+\frac{1}{R}}{(2-\phi_m)(r_{do}+s)} \tag{7.4}$$

where ϕ_m is the traffic skewness of the medium size flows. Setting k_s to some constant value, $k^*_{da} + k^*_{do}$ is known, and k^*_{da} can be computed from Equation 7.4. The key observation behind these expressions (that is also used to derive the optimal throughput) is that Cerberus achieves optimal throughput when the sub-topologies achieve the same DCT for a given load and their assigned traffic. That is, the utilization of the sub-topologies is balanced [3].

7.3.4 Initial Evaluation

In order to evaluate Cerberus, this section presents results obtained from flow-level simulations. The section first introduces the methodology and gives details on the compared configurations and then illustrates the results.

7.3.4.1 Methodology

This initial evaluation uses a custom event-based flow-level simulator written in Python. In contrast to a packet-based simulator, a flow-level simulator assigns end-to-end rates to flows. No packet buffering or switch processing latency occurs.

The considered network setup consists of $n = 64$ ToRs, $k = 16$ spine switches, $r_{do} = 10\,\mu s$, $r_{da} = 15\,ms$, $s = 90\,\mu s$ and a rate $R = 40\,$Gbps. The generated traffic corresponds to 16 hosts per ToR (1024 hosts in total) with 40 Gbps uplinks. The total host uplink capacity of the topology is 40.96 Tbps, which is at a comparable scale as other studies such as Sirius [53] (51.2 Tbps), Opera [49] (51.84 Tbps), or MegaSwitch [54] (63.36 Tbps). MegaSwitch's and Sirius' simulations operate in a similar range as this study; however, their implementations are not publicly available. In addition, the number of flows per simulation (200K with a mean flow size of 100KB) indicates that only very short periods, $< 1\,s$, were evaluated for Sirius. The main evaluation of Opera focuses on a scenario with much smaller topologies (only 6.48 Tbps) and lower load levels than this chapter. For larger scales [49, Figure 12], the details regarding flow generation are missing so that they cannot be judged.

The complexity of the *centralized rotor-net* implementation prohibits scaling to larger networks. The allocation of indirect traffic (2-hop routing) requires evaluating each source-destination rack pair. The scale of this operation is quadratic in the number of ToRs and executed for every slot. An optimized, centralized *rotor-net* algorithm is left for future work. For brevity, only the algorithms used in the simulation are described in the following.

7.3.4.2 Flow Assignment Algorithms

Three systems are compared: *expander-net*, *rotor-net* and Cerberus:
expander-net. *expander-net* uses a greedy single path routing on the expander graph built from the union of k matchings. First, all flows in the system are grouped by their source-destination ToR pair. For each pair of ToRs, the flows are sorted by their size in increasing order. The algorithm then iterates over the pairs and tries to allocate greedily as many flows as possible.
rotor-net. In contrast to the original distributed *rotor-net* algorithm implementation, the flow-based algorithm implementation uses a centralized control plane. It performs flow allocations on a ToR-by-ToR and switch-by-switch basis with a fixed, deterministic order. More advanced approaches are left for future work.
Cerberus. Generally, Cerberus uses the algorithms *rotor-net* and *expander-net* for each dedicated topology part. Following Algorithm 6, the algorithm first greedily searches for available circuits on the Da topology for the large flows. If no path can be found, the algorithm tries to allocate large flows on the Dobl part. Otherwise, the flows have to wait until capacity becomes available, in which case the algorithm tries again to (1) allocate flows on the Da or (2) on the Dobl part.

Note that these algorithms do not obtain optimal solutions, e.g., in contrast to solving a multi-commodity flow problem [294] or upper bounds described before. Rather they trade off the tractability of simulations at the desired scale and performance.

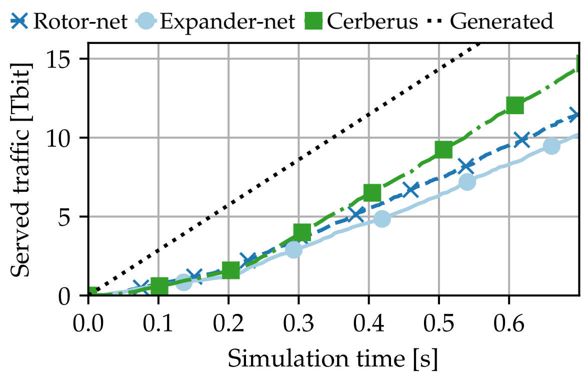
7.3.4.3 Traffic and Flow Size Distribution

Traffic is generated for an online scenario with the traffic model described in Section 7.3.3 using the Datamining [193] flow size distribution. The flow size distribution has an average size of 7.86 MB and $\phi = 0.45$. All ToRs are active ($x = 1$). The resulting flow size threshold for large flows is $t_l \geq 103.85$ MB which leads ≈ 79% of the bytes coming from large flows. The inter-arrival times of the flows follow an exponential distribution with mean values depending on the loads. The smaller the inter-arrival times, the higher the load. Note that the algorithms do not know when flows will arrive but have perfect knowledge of the flow sizes.

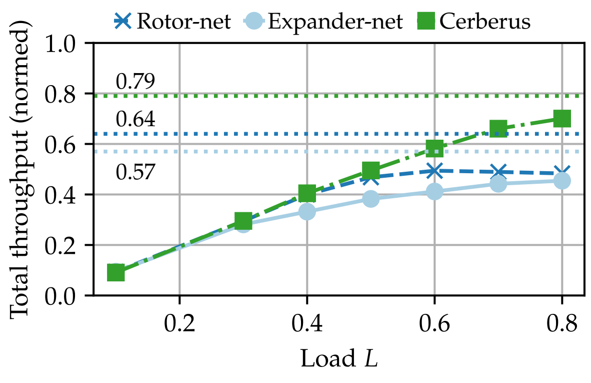
7.3.4.4 Results

Given the parameters setup of the simulation, the (maximum) throughput of each system can be computed from the respective expressions is: it is 0.79 for Cerberus, 0.64 for *rotor-net*, and 0.57 for *expander-net*.

In order to validate this result, Figure 7.2a presents the *accumulated* traffic (in bits) served by each system as a function of the simulation time for traffic generated for load $L = 0.7$. The dashed line illustrates the total incoming (or generated) traffic. From the analytical expressions, Cerberus



(a) Accumulated served traffic over time ($L = 0.7$).



(b) Throughput against load.

Figure 7.2 Served traffic over time (a) and throughput (b) from flow-level simulations for DATAMINING [193]. In (a), the dotted lines represent the arrived traffic and in (b), the maximum throughput according to the analytical expressions. CERBERUS outperforms *expander-net* and *rotor-net*.

should be able to support this rate while *rotor-net* and *expander-net* do not. We can observe that initially, all systems are unstable as flows start arriving. Around $t = 0.3$s, the systems become more stable and are able to process the incoming traffic at a constant rate. This processing rate, or the slope of the accumulated served traffic, shows the system's throughput for the given traffic load.

For $t > 0.3$s, the lines representing CERBERUS (green, squares) and the arriving traffic (dotted) are parallel. CERBERUS is able to support this load and maintains the same rate (slope) as the incoming traffic. This means that the *unserved* traffic volume, i.e., the distance between the lines for the offered traffic and CERBERUS, stays constant. Moreover, it means that the achievable throughput of CERBERUS is *above* 0.7, as we analytically calculated above. Note that the specific values and a deeper analysis of the unserved traffic are out of scope here since, to some extent, they depend on the simulation model and the implementation. In contrast, the rates (slopes) of *rotor-net* and *expander-net* are lower than 0.7, which means that the *unserved* traffic volume grows infinitely with time, and their throughput is lower than 0.7, which we also observed from the analytical expressions.

Figure 7.2b extends Figure 7.2a and shows the normalized throughput of each system (i.e., the processing rate of incoming traffic) for *different* traffic loads (solid lines). CERBERUS achieves higher throughput for higher loads compared to the other systems. For loads from $L = 0.1$ to 0.4, there is no significant difference between CERBERUS and *rotor-net* since both systems can support the incoming load. Only for loads higher than 0.5, CERBERUS achieves higher throughput than *rotor-net*: for instance, for $L = 0.6$, CERBERUS achieves 0.1 higher throughput (normalized unit), which translates to 4 Tbps higher throughput (i.e., $0.1 \cdot 16 \cdot 40\,\text{Gbps} \cdot 64$).

Additionally, the figure shows the analytical bounds for each system's (maximum) throughput as computed from the equations above (dotted lines). CERBERUS, *rotor-net* and *expander-net* have a throughput of $0.79, 0.64$ and 0.57, respectively. It is important to note that these bounds are computed for the accumulated demand matrix, so they present an upper bound for the online traffic generation process. Nevertheless, all systems approach the theoretical bound, preserving the rank between them and the relative improvements.

The simulation results also confirm the benefit of having *DA* topology components. This benefit becomes visible especially at higher loads, which are an interesting operational region for datacenter operators aiming to efficiently utilize their infrastructure resources.

7.3.5 Takeaways

Cerberus is a RDCN that combines multiple classes of reconfigurations, namely *DA* and *DO*, as well as integrates a static topology part for latency sensitive flows. Whereas the analytical results from prior work and the simulative evaluation in this section demonstrate its superiority over existing RDCN designs, the presented concept is still in an early stage. The next section elaborates more in detail on potential implementation and improvements of the individual sub-topologies.

7.4 Trio: An integrated, multi-class reconfigurable DCN

After having understood and evaluated, the general idea of Cerberus, this section presents Trio, a more in-depth end-to-end design. Trio employs the three-headed structure of Cerberus and uses the de Bruijn graph-based static topology from Duo (Chapter 6) that already allows easy integration of *DA* links. Moreover, Trio features a simple, rack-local packet scheduling for the *DO* topology part. Compared to the schedulers from *RotorNet*, Opera or Sirius, the presented scheduler does not require high synchronization effort between ToRs before sending traffic indirectly. Instead, it relies on the static and *DA* topology part to relieve temporal overload from ToRs. This offloading is enabled by a label-based source routing scheme which does not require updating forwarding entries on the ToRs for offloading traffic. Moreover, the source routing scheme prepares Trio for runtime adjustments of the sub-topologies' sizes — adding a new dimension of demand-awareness.

This section first presents the overall architecture, the simplified scheduler, and the data and control plane design (all Section 7.4.1). It then evaluates Trio using packet-level simulations on a range of traffic configurations (Section 7.4.2). Since the core idea is the same as in Cerberus, the evaluation compares Trio against two extreme points of the topology configuration: Duo and Opera. Moreover, it particularly focuses on the impact of different traffic patterns to highlight the benefits of the integrated design and demonstrate the superiority of matching traffic and topology.

7.4.1 The System Design of Trio

This subsection first presents the general architecture and topology components. Then, it describes in more detail the de Bruijn-based Da sub-topology and the packet scheduler for the Dobl sub-topology. The last part illustrates how integrated forwarding can be implemented in practice and presents the transport layer of Trio, supporting its fast reconfigurations.

7.4.1.1 Architecture

Figure 7.3 overviews Trio's architecture. Trio relies on a two-layer leaf-spine optical topology that can be described with the TMT model (cf. Section 3.6). The leaf layer consists of n ToR switches T_i ($1 \leq i \leq n$). Each ToR has k up- and k downlink ports of rate r. The latter ones connect to end-hosts so that there are $h = n \cdot k$ hosts in total in the network. Since the uplinks

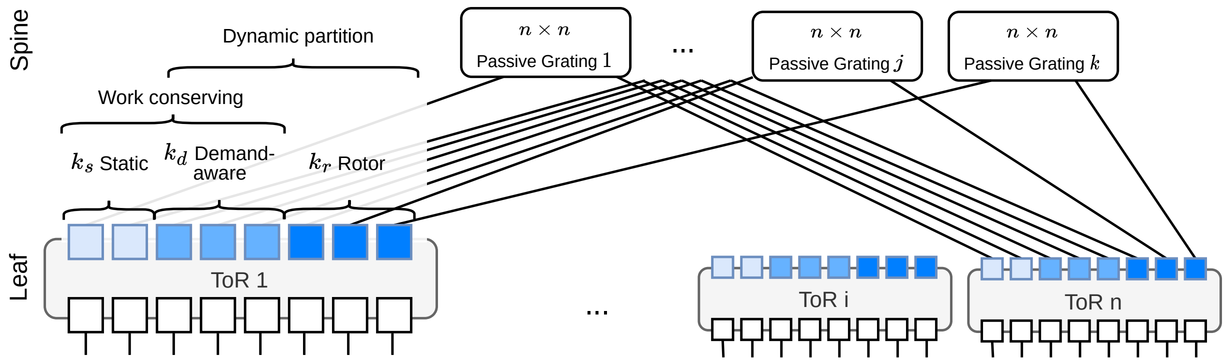


Figure 7.3 Overview of TRIO's architecture. The solid lines indicate bidirectional links. ToR switches connect in a two layer leaf-spine topology to AWGRs. A tuneable laser at the transceivers can adjust the wavelength to select an egress port. There are three laser scheduler classes (STATIC, DA, DOBL) that ports at the ToRs can be assigned to.

are bi-directional, we can further separate them into k unidirectional ingress and k egress ports. In contrast to CERBERUS and the original TMT model with k active optical spine switches, the spine layer consists of k passive gratings connecting to the ports at the ToRs. The reconfiguration mechanism works as proposed and demonstrated in Sirius [53]: Tunable lasers in each port of the ToR switches adjust the laser wavelength to select the egress port of the grating and, by that, change a (logical) topology link. Properly configuring the laser wavelengths at each time slot creates a directed multi-hop topology that is based on a set of k-directed matchings between the ToRs, i.e., the i-th ports in the ToRs are connected to the i-th grating, creating the i-th matching. Abstractly, this is exactly the structure as proposed in the TMT model.

In turn, the k matchings are partitioned into three *link scheduling* classes[6]: STATIC, DOBL, and DA. The sizes of the classes are k_s, k_{do}, k_{da}, respectively, keeping the constraint $k = k_s + k_{do} + k_{da}$. A partition to three classes is implemented by assigning the k ports in each ToR to one of the three link schedulers (STATIC, DOBL, and DA) in a consistent and symmetric way. In all ToR switches, the same k_s ports use the STATIC link scheduler, the same k_{do} ports use the DOBL link scheduler, and the same k_{da} ports use the DOBL link scheduler. Each different link scheduling class creates a different sub-topology, and the three sub-topologies are described in the following:

- **Static sub-topology (STATIC):** The k_s static ports do not reconfigure the links over time. The resulting topology can be described as the union of k_s static matchings. The static ports provide basic connectivity between the ToRs. They can be used to create regular graphs, such as expander graphs, providing low latency for short flows using multi-hop routing. Specifically, TRIO relies on de Bruijn graphs [304] for the STATIC topology component, which are described in more detail in Section 6.2.

- **Demand-aware sub-topology (DA):** The *DA* topology consists of a collection of k_{da} DA reconfigurable ports per ToR. They can flexibly be reconfigured to *any* possible k_{da}-regular directed graph between the ToRs and change it over time. For example, these ports can be used to create direct connections between ToR pairs with high communication demand. The laser's pure reconfiguration delay is in the order of nanoseconds [53]. However, reconfiguring the DA ports requires coordination between the ToRs (e.g., for data collection and decision-making).

[6]Denoted as spine switch types in the TMT model.

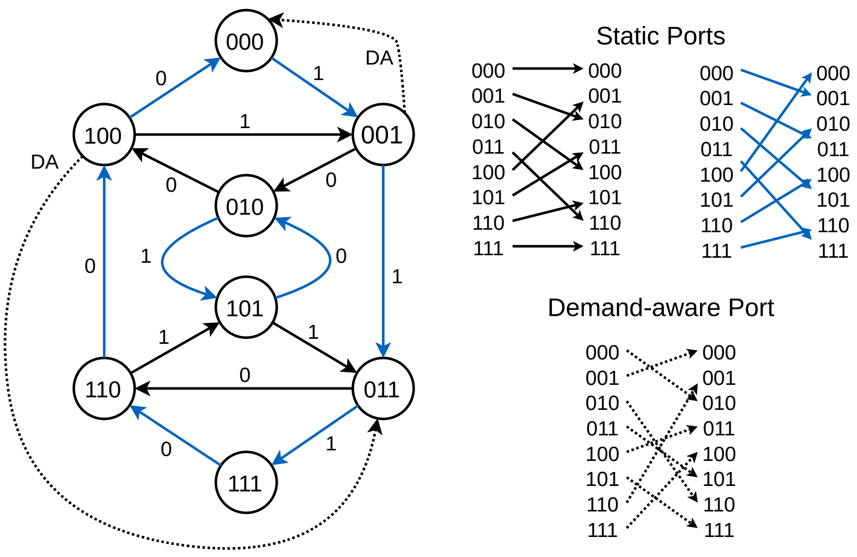
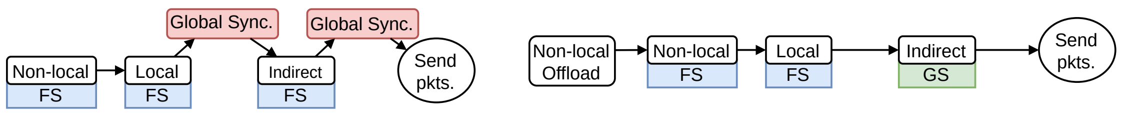


Figure 7.4 An example of TRIO's backbone sub-topology with eight ToRs: A de Bruijn topology established by two STATIC ports with and an additional DA port. The right half shows the two matchings for the STATIC ports and the additional matching of the DA ports (for clarity, drawing only two links in the topology).

The *reconfiguration delay* of a DA port used in this chapter and denoted by r_{da}, accounts for this. The default value is $r_{da} = 1ms$, but the evaluation in Section 7.4.2 also considers other values. The circuit-hold time after each reconfiguration is dynamic (not constant) during the operation of a DA port. As a rule of thumb, it should be much larger than r_{da} for the reconfiguration to be worthwhile. Otherwise, most of the time is spent on reconfigurations, and the utilization of the link would be small.

- **Demand-oblivious sub-topology (DOBL):** The last sub-topology is formed by the set of k_{do} dynamic, DOBL ports per switch. Similar to the original rotor switches [48], each DOBL port cycles through $n-1$ predefined matchings, emulating a fully-connected network (i.e., complete graph) over time. Each one of the k_{do} DOBL ports cycles through the same $n-1$ matchings but using a different time offset. This synchronization provides an average *cycle time* of $\frac{n-1}{k_{do}}$ slots to complete a single emulation of a complete graph between all ToRs. The symmetry between the ports' link scheduling enables a flexible way to add/remove DOBL ports since the only difference between the ports is the time offset. The slot time of the DOBL link scheduling class is defined by a circuit-hold time, denoted as s, plus a reconfiguration delay denoted as r_{do}, which includes the physical link reconfiguration (wavelength change) and additional quiet time to empty active links. The *duty cycle* is the fraction of the time traffic can be sent in a slot (i.e., $\frac{s}{s+r_{do}}$). The slot time is tuneable and depends on r_{do}, where a reasonable setup is to achieve a duty cycle above 90% as in [48, 53]. This chapter assumes a duty cycle of about 98%, as in [49].

Since the partition of sub-topologies is determined by the link scheduling classes of the ports but not by the underlying infrastructure, the assignment of ports to link scheduling classes can change over time and, by that, change the sub-topologies' sizes. In contrast to CERBERUS (Section 7.3), this adds a new dimension of demand-awareness to TRIO where k_s, k_{da}, and k_{do} can dynamically adapt. For instance, a DA port can become a DOBL port if the traffic pattern has changed such that the DOBL sub-topology component is under-provisioned. While this feature can generally be



(a) RotorLB

(b) LocalLB

Figure 7.5 High-level comparison of Opera/RotorNet (RotorLB) and Trio packet scheduling (LocalLB) on the Dobl topology. LocalLB relies on offloading the non-local traffic to the Static and Da sub-topologies to remove the need for a global synchronization.

used for all three classes, Trio limits the *dynamic partitioning* of the topology to the Da and Dobl topology components.

The three sub-topologies have different properties for the packet forwarding behavior: For example, Static and Da can utilize work-conserving forwarding (in the sense of store and forward), whereas for Dobl to transmit packets successfully, it requires a packet scheduling algorithm to handle the periodic link changes. Both parts are introduced in the subsequent sections.

7.4.1.2 Static and Da Topology

In order to implement the topology part with a work-conserving forwarding, [7] Trio augments a de Bruijn graph-based static topology built from the Static ports with dynamic connections (short-cuts) on the Da ports. It closely re-uses the concepts from Duo (Chapter 6). In particular, and in contrast to previous proposals [47] and also Cerberus, Duo supports *integrated multi-hop* routing across links from both the Static and Da topology parts, i.e., a single packet can traverse both sub-topologies to reach the destination. Moreover, the structural properties of the de Bruijn-graph allow leveraging (IP-based) *greedy* routing using small forwarding tables. Lastly, the update cost of a forwarding table upon a Da link change is small and can be performed locally with the new neighbors.

Figure 7.4 shows an example of such a hybrid de Bruijn topology with eight ToRs, two Static ports per ToR, and one Da port. The node IDs are in binary representation. The topology is, therefore, a union of three matchings, two Static and demand-oblivious, and one dynamic and Da. The matchings are shown in the right half of the figure. For details on this sub-topology, the reader is referred to Chapter 6, which presents Duo. While the Duo topology has many benefits that are also implemented in Trio, it does not have a Dobl sub-topology which the next part discusses.

7.4.1.3 Packet Scheduler for Dobl Topology

As in previous work [53, 48, 49], a non-work-conserving packet scheduling algorithm is needed to transmit packets via the highly dynamic Dobl sub-topology successfully. Recall that in this sub-topology, links between ToRs are constantly and systematically changing, emulating a complete graph between the ToRs over time while being oblivious to the demand. Therefore, packets potentially need to be stored while waiting for their next-hop link to be reconfigured. In Trio, similar to previous work, packets are buffered at the end hosts when they need to be delayed (and not at the ToR switch) and forwarded to the ToR switch based on a ToR-host synchronization and the packet scheduler.

[7]For the definition of "work-conserving" in this context please refer to Section 6.1.1

Algorithm 7 TRIO Offloading (from DOBL to STATIC & DA) at host v

```
1: for dest $u = 1 \dots h$, ($v$ and $u$ not in same rack) do
2:   if $D^l_{v,u} > \frac{C}{k}$ then
3:     Offload $D^n_{v,u}$ to backbone topology
4:   else
5:     if $D^l_{v,u} + D^n_{v,u} > \frac{C}{k}$ then
6:       Offload $D^l_{v,u} + D^n_{v,u} - \frac{C}{k}$ from $D^n_{v,u}$ to backbone
```

The state-of-the-art approach for this challenging task is the RotorLB (RLB) scheduling proposed with RotorNet and Opera [53, 49]. It considers both *direct* (single hop) and *indirect* (of at most two-hops) routes and uses Valiant-based routing [171] (via a random intermediate helper node) to achieve load balancing.[8] This approach was shown to provide high throughput. RLB has for each host a set of *virtual buffers* for each destination, both for *local* traffic that is generated by the host and for *non-local* traffic where the host acts as an intermediate node. Unfortunately, RLB introduces a significant control plane overhead, in particular since intermediate buffers cannot easily handle overflows, and a tight sender-receiver flow control mechanism needs to be implemented. At the beginning of every slot, hosts (in different racks) negotiate the amount of traffic that can be sent indirectly to prevent overflow. In contrast, TRIO implements a simpler packet scheduler (for the DOBL topology), LocalLB (LLB), that does not require such global synchronization. The decisions of what to send indirectly are made (rack-)locally. In case of an overload of a specific rack/host, i.e., if non-local traffic accumulates, TRIO can rely on its efficient backbone topology (STATIC and DA) to forward long waiting traffic to the destination. A second difference is the capacity per source that is reserved. OPERA assumes a uniform distribution of the traffic and, therefore, a priori, applies an equal share of the slot capacity across all (source) hosts in a rack. The initial sending capacity assigned to each host is $\frac{C}{h}$ where C is the slot capacity and h is the number of hosts. TRIO does not make such an assumption but allows a more flexible distribution of the resources within a rack.

High-level Scheduling Process. Figure 7.5 visualizes the scheduling process at the beginning of a slot for OPERA and TRIO. Table 7.2 summarizes the used notation of constant values. Table 7.3 defines the input (sizes of the local and non-local buffers) and output data structures (allocated volumes from local and non-local to the destination and from local to an intermediate host).

First, each host in TRIO checks if non-local traffic demand must be offloaded from the DOBL sub-topology to the backbone sub-topology (which supports work-conserving forwarding, and the traffic can immediately be sent to the ToR switch). The offloading (Algorithm 7) follows a simple, greedy logic. Each host v checks for each destination u (that is not in the same rack) whether the local demand $D^l_{v,u}$ exceeds what can be sent in a single slot and a single path, assuming all-to-all traffic and fair-share, i.e., $\frac{C}{k}$. If so, all the non-local traffic to u, $D^n_{v,u}$ is offloaded to the de Bruijn-based backbone. If the local demand to u fits in a single slot, the algorithm compares the total demand from v to u. If it exceeds the fair-share capacity, the excess demand is offloaded (from the non-local traffic) to the backbone. Note that TRIO offloads only non-local demand to the backbone.

[8]For short, time-sensitive packets, Opera [49] uses longer multi-hop paths routing, but these packets capture only a small fraction of the total traffic.

Table 7.2 Constants used by TRIO's packet scheduling.

Variable	Description
n	Total number of ToRs.
k	Total number of spines
k_{do}	Total number of DOBL spines
T_i	ToR i, $1 \le i \le n$
R_i	DOBL switch i, $1 \le i \le k_{do}$
$h = n \cdot k$	Total number of hosts
$C = r \cdot \delta$	Slot active capacity (volume to send)
r	Link rate
$\bar{C} = \frac{C \cdot k_{do}}{k}$	host ToR DOBL switch capacity
$h_{i,j}$	Host i on ToR j (can be translated to an integer for indexing)
$u_{i,j}$	Uplink i on ToR j, $(1 \le i \le k_{do})$
$T_{u_{i,j}}$	ToR connected on uplink i on ToR j

Table 7.3 Input and output data structures of TRIO's packet scheduling.

Variable	Description
Input	
$D^l_{i,j}$	Current local demand from host i to host j.
$D^n_{i,j}$	Current non-local demand from host i to host j.
Output	
$S^{ld}_{i,j}$	Direct local volume to send from host i to j.
$S^n_{i,j}$	Non-local volume to send from host i to j.
$S^{li}_{i,j,k}$	Indirect local volume to send from host i to j with final destination k.

Next, the non-local demand is scheduled. The procedure is the same for OPERA and TRIO, using a fair-share (FS) algorithm over source and destination hosts. The same procedure is applied for local direct demand. After this step, OPERA creates offers to be exchanged between connected ToRs, thereby synchronizing demand information across the network to perform flow control (red). In TRIO, this step is not necessary, saving complexity. For allocating new indirect traffic, OPERA uses a fair-share approach again to distribute the remaining capacity across the source hosts and, in particular, the destinations per host. In contrast, TRIO follows a more greedy approach since it can handle buffer overloads (GS, see next paragraph). Each host selects the destination with the largest demand and then distributes the remaining capacity in the slot to all hosts with remaining local demand again greedily: The source hosts within a rack are sorted by their largest destination, and capacity is assigned starting with the smallest. If the demand of the currently considered source host exceeds the remaining slot capacity, the capacity is equally shared among all hosts with demand. Otherwise, it is assigned as much as needed. The process is repeated until all demands are served or the remaining capacity has been used.

Detailed Algorithm. This section provides a detailed description of the algorithms underlying TRIO's packet scheduling. Algorithm 8 is a sub-routine used by LLB. The routine is summarized in pseudo-code from prior work (the code base) from OPERA [49]. Given a matrix I of demand to be sent, it applies a 2-dimensional fair share over the sending and receiving capacities (given as vectors). Therefore, it repeats sweeping the rows (l. 5f) and then the columns (l. 7f) until the output values converge.

Algorithm 9 is the whole packet scheduling algorithm with the major steps as described before. It takes a per ToR perspective (the algorithm runs on ToR t); all data structures are rack-local, but the algorithm listing partially uses global indices to simplify notation.

In lines 3-7, the algorithm allocates the second hop non-local traffic and local traffic that can be sent directly to the destination. For each buffer type (non-local (n) and local (l)), it iterates over the DOBL ports and collects the per-port demand information (local or non-local respectively) of all source and destination hosts that are connected by the port. Then, it allocates the demand using the 2-dimensional fairshare (FS2D, l. 6). The second part (lines 9-25) allocates traffic to be sent

Algorithm 8 2-dimensional fair share (from OPERA's implementation [49]).

```
1: $I \in \mathbf{R}^{N \times M}, c0 \in \mathbf{R}^{N}, c1 \in \mathbf{R}^{M}$
2: Initialize output: $O \in \mathbf{R}^{N \times M}$
3: Count elements $> 0$ in $I$ as $n^+$
4: while $n^+ > 0$ and max iterations not reached do
5:     for $i = 1 \dots N$ do
6:         $t[i]$ = 1-dimensional fair share on $I[i]$ with capacity $c0[i] - \sum_{j=1 \dots M} O[i,j]$
7:     for $j = 1 \dots M$ do
8:         $t2$ = 1-dimensional fair share on $j$-th column of $t$ with capacity $c1[j] - \sum_{i=1 \dots N} O[i,j]$
9:     Update $t$ with $t2$
10:    Update $I, O$ with $t$
11:    Zero out rows $I[i]$ where $c0[i] = 0$
12:    Zero out columns $I[j]$ where $c1[j] = 0$
13:    Count elements $> 0$ in $I$ as $n^+$
```

via an intermediate host. Again, the algorithm iterates over the DOBL ports. Per port, it iterates over the hosts in the destination rack (with the order changing in a round-robin fashion between calls of the packet scheduling). For each such candidate intermediate host, it searches for the destination with the highest demand per source host (lines 13-17). Then, it iterates over these demands in non-decreasing order. If a demand z_i exceeds the remaining capacity in the slot to the intermediate host, the equal-share of the remaining capacity is given to all remaining demands > 0 (lines 19-22). If the demand does not exceed the remaining capacity to the intermediate host, it is greedily allocated (l. 20f). Note that the algorithm does not evaluate the capacity of the slot from the intermediate node to the actual destination (the second hop) but relies on TRIO's offloading (Algorithm 7) to compensate for overloads.

7.4.1.4 Technical & Implementation Details

TRIO's architecture with three sub-topologies comes with several challenges regarding integration and synchronization. This section presents more details on the implementation of TRIO. It first covers flow classification and feasible transport protocols and components of data and control plane. Afterward, it describes how packets are forwarded and the synchronization between hosts and ToRs.

Flow classification. TRIO requires a mechanism to classify flows and to separate traffic onto the three sub-topologies. One option is to estimate the flows' sizes to classify them as done in OPERA [49], CERBERUS [3], and DUO. TRIO uses this method to distinguish STATIC and DA traffic, for instance, using approaches such as flow aging communicated from the application. However, the DOBL sub-topology is particularly suited for *uniform* traffic patterns, such as flows that belong to the shuffle phase of a map-reduce job. Therefore, TRIO additionally relies on application-level information for traffic classification. That is, either a shim layer between the application and the network stack on the host is added to insert additional tags to the packet headers identifying DOBL traffic, or alternatively, transport layer ports are used to identify the applications with uniform communication patterns and mark DOBL traffic accordingly.

Algorithm 9 TRIO Packet-Scheduling (LocalLB) for ToR t

```
1:  $C_i^{TX} \leftarrow \bar{C}, C_{i,b}^{RX} \leftarrow \frac{C}{k}, \forall i = 1 \dots k, \forall b = 1 \dots k_{do}$
2:  //Allocate Non-local (n) on the 2nd hop and local (l)
3:  for x ∈ $\{n, l\}$ do
4:    for $b = 1 \dots k_{do}$ do
5:      $W_{i,j} \leftarrow D^x_{h_{i,t}, h_{j,T_{b,t}}}$  $\forall i = 1 \dots k, j = 1 \dots k$
6:      $S^x \leftarrow FS2D(W, C^{TX}, C^{RX})$
7:      Update $C^{RX}, C^{TX}, D^x$
8:  // Allocate new indirect traffic on the remaining capacity
9:  for $b = 1 \dots k_{do}$ do
10:   // Iterate over all hosts in ToR connected via $b$
11:   for $\forall a \in \{h_{j,T_{b,t}}, j = 1 \dots k\}$, order RR over slots do
12:     //Find largest demand per source host capped by remaining sending
        capacity
13:     $V_{i,m} \leftarrow D^l_{h_{i,t}, h_m} \forall i = 1 \dots k, m = 1 \dots nk$
14:     $v_{i,m} = v_{i,m} - \frac{C}{k}$ and zero all $< 0$
15:     $y_i \leftarrow \arg\max_m v_{i,m}, \forall i = 1 \dots k$
16:     $z_i \leftarrow \min\{v_{i,y_i}, C_i^{TX}\}$
17:     $n_z \leftarrow$ number of $z_i > 0$
18:     for $\forall i$ where $z_i > 0$ in non-decreasing order of $z_i$ do
19:       if $z_i > C_{a,b}^{RX}$ then
20:         Allocate $\frac{C_{a,b}^{RX}}{n_z}$ for all $i$ with $z_i > 0$
21:         Update $C^{TX}, C^{RX}, D^l$
22:         Break
23:       else
24:         Allocate $z_i$ for $i$ and update $C^{TX}, C^{RX}, D^l$
25:         Decrement $n_z$
```

Transport protocols. TRIO uses different transport protocols for the three sub-topologies. Latency-sensitive (small) flows run via the STATIC topology with NDP [172], which has shown good performance for such traffic. Flows that are transmitted via the DOBL part are sent according to the schedules determined by LLB (Section 7.4.1.3). Throughput-sensitive flows use standard TCP for transmission on the STATIC and DA ports to efficiently share the available resources.

Data and Control Plane Components. TRIO uses label-based source routing and priority queueing to forward the flows onto the different topology classes and to turn the decisions of LLB into action. Figure 7.6 visualizes the involved components on the hosts and the ToRs. The example considers a TRIO configuration with one DOBL (R0), one DA (D0), and two STATIC ports (S0, S1) on the ToR switch. All three flow classes share the up-link from the host to the ToR. In order to reduce interference, TRIO uses priority queues on hosts and ToR switches. STATIC flows are given the highest priority, followed by DOBL and DA. Note that queues for STATIC and DA traffic are not needed on DOBL ToR uplink ports, but the queue for DOBL traffic is needed on both STATIC and

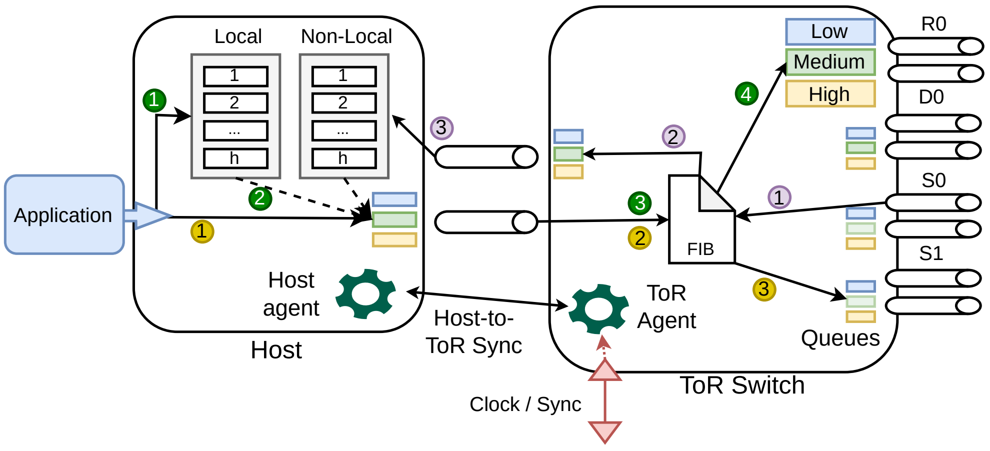


Figure 7.6 Overview of TRIO's data- and control plane components on hosts and ToRs. Each link has three queues of different priorities to separate traffic for the three sub-topologies. Packets from the application are tagged with a label. The FIB matches on this label to identify the sub-topology or active slot on the *DO* topology. Host and ToR agent synchronize the demand and assigned resources as well as the active slot on the DOBL sub-topology.

DA ports to enable the *offload* from DOBL topology to the de Bruijn backbone. Besides the port queues, four more entities are involved:

- The *Local* and *Non-local* buffers store packets for transmission via the DOBL part. Similar to RotorNet and Opera, each of them features a dedicated virtual queue per destination in the network [48, 49].
- `Host agent` runs on every host and sends the individual sizes of the *Local* and *Non-local* buffers to a rack-local coordinator (`ToR agent`). It waits for *pull* messages from the `ToR agent` to send packets from the buffers. The pull messages contain the volume to be sent per destination. When sending a packet for the DOBL topology, the host adds a *slot label* that either indicates the active DOBL matching or is 0 if the packet is to be sent via STATIC or DA ports.[9]
- `ToR agent` coordinates the DOBL packet transmission of all hosts in a rack. It can run on of the ToRs or on one of the hosts. It receives demand information from the `Host agents` and runs LLB. The outcome is sent to the hosts along with the slot label to be used (*pull* messages).
- `FIB` is a single forwarding table on the ToR for all three sub-topologies. Besides the destination IP address, the `FIB` matches the slot label to obtain the egress port. Thereby, forwarding to DOBL or DA and STATIC can be differentiated. The entries for the DOBL links can be pre-computed and do not change over time (unless the number of DOBL ports changes). Forwarding entries for DA and STATIC might be updated if the DA links change.

The following part describes how these components interact and synchronize to effectively forward packets over the three sub-topologies.

Packet forwarding. Figure 7.6 illustrates the major steps taken when forwarding packets in TRIO. Packets from applications that should be sent via the DOBL are put to the corresponding *Local*

[9]Every DOBL port cycles through $n-1$ configurations. The slot label indicates the currently active configuration. At every point in time, all hosts use the same slot label.

destination queue ①. If packets are offloaded from DOBL to the STATIC and DA sub-topologies, the `Host agent` takes packets from the queue and directly sends them ②. In this case, the 0 slot label is added to the packet. Otherwise, the packets are sent according to the calculated schedule (cf. Section 7.4.1.3) with the slot label as provided by the `ToR agent`. Packets that are sent indirectly via the DOBL part are encapsulated with the address of the intermediate host. On the ToR, the packet is matched in the FIB ③ and forwarded using the medium priority queue ④.

Packets from applications to be sent via STATIC or DA (Non-Rotor) are tagged with slot label 0 and then sent out to the ToR ①. Here, they are again matched in the FIB ② and forwarded accordingly ③. Traffic received on the ToR's uplinks is forwarded to the destination host ① ②. When a host receives indirect DOBL traffic, it adds the packets to the corresponding *non-local* queue ③. Packets received on the final destination (host) are forwarded to the application. On the ToR, all packets are matched in the FIB, the slot label is popped, and the packet is sent to the egress port.

FIB Example. To illustrate the content of ToRs' FIBs, we consider a network with eight ToRs. We consider a point in time at which STATIC and DA ports are connected according to the scenario depicted in Figure 7.4. In addition to the ports shown there, each ToR has one DOBL port. The IP addressing scheme in this example follows the description in Section 6.2.3.2. Bits 22-24 encode the de Bruijn node addresses. The lower 21 bits can be used to address hosts inside the rack. The rules match on a combination of exact match (for the slot label) and longest prefix match (LPM) (for the destination address). To give an example, Table 7.4 shows the FIB for node 100. The upper part shows the forwarding rules for the STATIC and DA part. They essentially match the forwarding rules from DUO (cf. Section 6.2) .[10] The only addition is an exact match on the slot label to be 0. The lower part of the FIB shows the rules for the DOBL forwarding. Here, the matching value of the slot label depends on the active slot. The match also considers the destination address to perform forwarding if multiple DOBL ports are present. Packets that do not match any rule are dropped.

DOBL Synchronization. TRIO requires two types of synchronization for the DOBL sub-topology. First, ToRs need to synchronize their configuration state (i.e., the slot) globally across the network. This can be achieved using a global (broadcast) clock signal (red triangle in Figure 7.6).

Second, TRIO needs synchronization between hosts and ToRs to put the decisions of LLB into effect. Therefore, TRIO considers a similar approach to RotorNet and Opera. Packets are primarily buffered on the hosts. Demand information is pushed *once* per slot from the `Host agent` to the `ToR agent`, e.g., with Remote Direct Memory Access (RDMA) messages [48]. After having calculated the number of packets to send with LLB, the `ToR agent` pulls traffic from the host, i.e., notifies the `Host agent` about how much to send. This can be done again using RDMA messages. Other approaches might be possible as well. For instance, for its rack-based deployment, Sirius can operate with the buffers on the ToRs to implement the needed local queues. The authors refer to Credit-based flow control mechanisms as available in InfiniBand to avoid buffer explosions on the ToR. However, Sirius negotiates the scheduling almost on a packet-by-packet basis. While this approach reduces the buffer requirements, it comes at the cost of high inter-ToR synchronization effort. Although TRIO comes with several challenges regarding the integration of the three sub-

[10]The rules are not sorted. Their order does not necessarily reflect the order in memory/the order of the lookup.

Table 7.4 Forwarding table on ToR 100 following the Static and Da topologies in Figure 7.4 with one additional Dobl port. Rows 1-7 forward on the de Bruijn sub-topology, rows 8-14 on the Dobl topology, and row 15 drops incorrectly tagged packets.

#	Slot Label	Dst. Addr.	Port
1	0	10.192.0.0/10	DA
2	0	10.160.0.0/11	$\{1, DA\}$
3	0	10.96.0.0/11	$\{1, DA\}$
4	0	10.64.0.0/11	1
5	0	10.32.0.0/11	1
6	0	10.0.0.0/11	0
7	0	10.128.0.0/11	Local
8	1	10.96.0.0/19	{Dobl }
9	2	10.128.0.0/19	{Dobl }
10	3	10.160.0.0/19	{Dobl }
11	4	10.192.0.0/19	{Dobl }
12	5	10.224.0.0/19	{Dobl }
13	6	10.0.0.0/19	{Dobl }
14	7	10.32.0.0/19	{Dobl }
15	*	*	drop

topologies, all the described aspects in this section illustrate the feasibility of Trio. Moreover, they describe an avenue toward an implementation that is partially explored and evaluated in the following section.

7.4.2 Empirical Evaluation

This section evaluates Trio with packet-level simulations using *htsim* [172]. It compares Trio to Opera and Duo, across a range of traffic patterns that build on available empirical flow size distributions and synthetic patterns. The evaluation focuses on goodput, which accounts for re-transmissions and re-ordering, as the main performance metric but also investigates flow completion times and the efficiency of the resource usage.

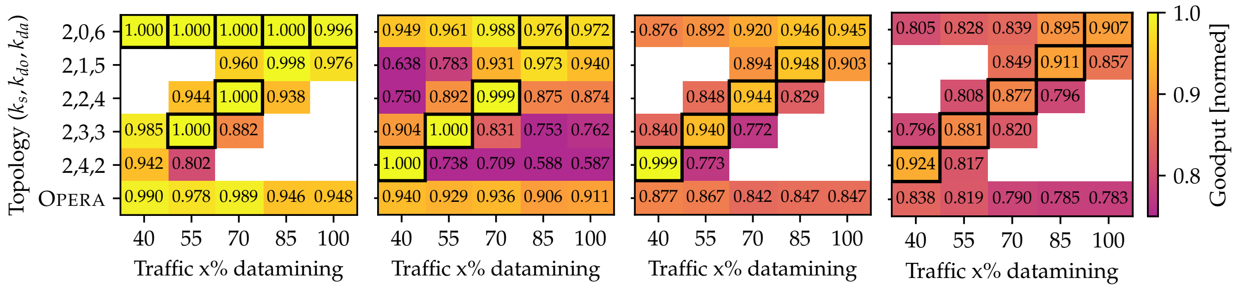
7.4.2.1 Settings

We consider topologies with $n = 64$ ToRs. Each ToR has 16 bi-directional ports which are equally split into up- and downlinks ($k = 8$). All links have a capacity of 10 Gbps. This results in a total uplink capacity of 5.12 Tbps, which is comparable to previous works [49, 53].

The reconfiguration periods are chosen such that the duty cycles in both dynamic topology parts are around 98%, similar to prior work [49]. Unless stated otherwise, the physical reconfiguration delay is 100 ns for Dobl switches and 1 ms for Da switches. On the Dobl part, there is a 1.7 µs guard period to empty the fibers before reconfiguration, similar to the settings in [49].

Topologies. The three systems are described in detail in the following:

- **Trio** uses the hybrid, multi reconfiguration-class topology presented in Section 7.4.1. Throughout the evaluation, the proportions of the sub-topologies are denoted by the tuple (k_s, k_{do}, k_{da}). Scheduling of the Da links is greedy according to the size of the demands using the centralized algorithm presented in Section 6.2.2. For both packet scheduling on the Dobl part and



(a) L = 35% **(b)** L = 40% **(c)** L = 45% **(d)** L = 50%

Figure 7.7 Goodput averaged over 1 s of simulation. The heatmap compares topology configurations and traffic mixes. Each cell shows the average of 10 simulation runs. The values are normalized to the offered traffic. Black rectangles indicate the best value per traffic mix (column). Since the goodput of the elements that are far away from the anti-diagonal in the heatmap are expected to be low (cf. Figure 7.7b), they are omitted for L = 35%, 45% and 50% (white cells). For loads ≥ 40%, adjusting the sizes of the sub-topologies to the respective traffic share increases the goodput.

scheduling of Da links, the system has full knowledge of the flows' sizes at any point in time; an assumption also made in prior work [49]. The allocation of flows to the scheduling classes (i.e., the sub-topologies) relies on two criteria. First, this evaluation assumes that there is application-level information to identify traffic for the Dobl part (see also the traffic description later). Second, a size threshold of 1 MB is used to separate traffic for the only Static part (using NDP) and traffic for the Da part (using TCP).

- **Duo** (Chapter 6) is a specific configuration of Trio with $k_{do} = 0$. As in Chapter 6, this evaluation considers a configuration with $k_s = 2$ and $k_{da} = 6$. The Da links are scheduled greedily based on the remaining demand volume between the ToR pairs Algorithm 3. The threshold for the Da link scheduling is 10 MB. The algorithm has full knowledge of the flows' sizes upon their arrival. The routing uses the properties of de Bruijn-based greedy routing, i.e., paths combining Static and Da links are possible. Similar to Trio, flows < 1 MB use NDP and larger flows use TCP as transport protocols. Queues in the two systems above can hold 50 data packets of size 1500 B.

- **Opera** [49] is a demand-oblivious, dynamic topology and serves as the second baseline. It periodically cycles through a set of matchings that are generated so that it maintains an expander graph at every time. Opera also splits the flows into low-latency and bulk traffic. Low-latency traffic is forwarded via the temporarily static expander part of the topology with NDP as the transport protocol. Bulk traffic is scheduled and sent with the RotorLB protocol and scheduling [48, 49]. The evaluation considers the default configuration for queue sizes (8 · 1500 B) and the threshold for bulk traffic of 15 MB.

Traffic. The evaluation considers an online traffic scenario where flows arrive over time according to a Poisson process. To demonstrate that Trio is particularly suited for traffic patterns that mix skewed demands with uniform patterns, generated traces contain traffic from two distributions and varying shares of the two traffic patterns (skewed and uniform). The variable x denotes the share of the skewed traffic. For the skewed traffic component, connection pairs are sampled uniformly at random. The flow sizes are sampled from distributions similar to Datamining [193],

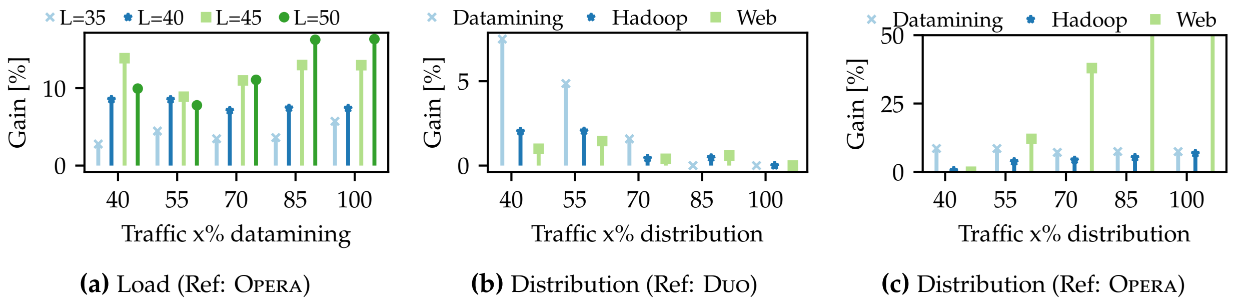


Figure 7.8 Gain in goodput when using Trio against the fraction of skewed traffic. The sub-figures compare different load values (a) and flowsize distributions for the skewed traffic (b & c). The reference topology is indicated in the sub-caption.

Hadoop [34], and WebSearch [311]. Specifically, the distributions have been super-sampled by a factor of 5 to increase the number of distinct flow sizes. The uniform pattern is composed of demand matrices with one flow of size 112.5 KB for each host pair in different racks. The flows of a matrix arrive over time, and the arrival rate of the generated matrices controls the share of skewed traffic. The load L is controlled via the arrival rates of the flows. We assume that all three systems can identify traffic belonging to the uniform traffic pattern using application-level information or special tags. Hence, they are configured so that they forward flows belonging to the uniform traffic via the Dobl sub-topology (regardless of the threshold for large flows). The results report on mean values of 10 runs per setting.

7.4.2.2 Goodput analysis

We start with evaluating the goodput of Trio in various scenarios.

A matched topology configuration increases goodput. Figure 7.7 visualizes the average goodput over 1 s of simulation for different topology configurations and traffic mixes. The values are normalized to the offered load in the respective run. A value = 1 means the topology can fully serve the offered traffic. The skewed traffic is sampled from Datamining. Comparing the different loads (sub-figures), we note that the normalized values decrease slightly with increasing load. For $L = 35\%$ (Figure 7.7a) the number of cells = 1 is highest, whereas for $L = 50\%$ (Figure 7.7d), none of the configurations can fully sustain the offered traffic. This is expected as the congestion in the topologies increases.

Looking at the individual heatmaps (for instance, Figure 7.7b), the performance of the topology configurations varies with the traffic mix. That is, for each share, there is one best configuration.[11] For instance, for $x = 70\%$ (70% Datamining and 30% uniform traffic), Trio with $(2, 2, 4)$ achieves the highest goodput, i.e., the normalized values are closest to 1. This effect is weaker for low load ($L = 35\%$). Moreover, topology configurations with a small number of Dobl links serve better traffic mixes with higher x. This aligns with the conclusions made in Section 7.3 (and more in detail in [3]). The share of Dobl links approximately corresponds to the share of the traffic in the uniform matrix. For instance, configuration $(2, 2, 4)$ dedicates 75% of the resources to Static and Da and performs best for $x = 70\%$,in which 70% of the traffic uses these sub-topologies.

[11] For demonstration, only the heatmap for $L = 40\%$ is complete. The white regions in the other heatmaps are similarly off from the offered load.

Here, the normalized goodput is 0.999, compared to 0.892 and 0.875 for $x = 55\%$ and $x = 85\%$ respectively. Overall, over-provisioning the Dobl component reduces the goodput more than over-provisioning the Da part.

Compared to Trio, Opera performs worse for all considered traffic mixes (and load levels). For instance, for $L = 40\%$, it achieves approximately 6% lower goodput than Trio. Duo, which resembles a special case of Trio (with $(2, 0, 6)$, performs the best for the high x. Here, the purely *DA* topology better matches the highly skewed traffic. Choosing the correct configuration for Trio is important for maximizing its performance. If not stated otherwise, the following analyses use $L = 40\%$ and $x = 70\%$ sampled from Datamining.

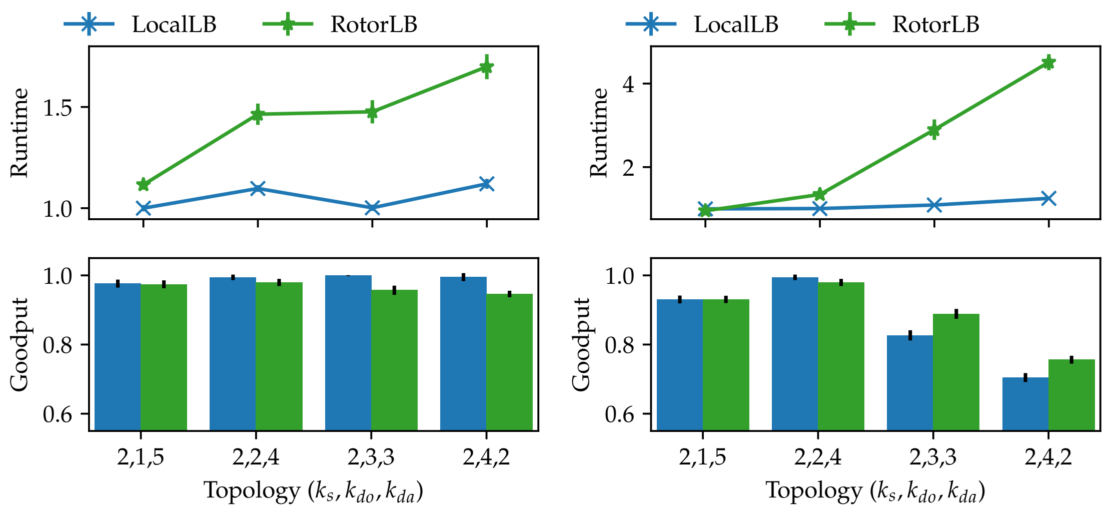
Trio's gains persist for other loads & other flow size distributions. Figure 7.8 summarizes the relative gain in goodput when using Trio compared to the indicated reference configuration. The gain is calculated as $\frac{G_{\text{Trio}} - G_{ref}}{G_{ref}}$. Note that $(2, 0, 6)$ (Duo) is also considered as part of Trio here. Figure 7.8a illustrates the gain against the share of skewed traffic for different loads. For $L = 35\%$, the gain varies between $0 - 4\%$ but does not show a clear trend. For $40\% \leq L \leq 50\%$, the behavior is different. For these load values and $40\% \leq x \leq 70\%$, the gain remains almost constant around 10% – neither the load nor the share of skewed traffic shows an impact here. The only exception is $L = 45\%$ and $x = 40\%$, with a gain of 14%. This value steps out since at $L = 45\%$ Opera fails to serve the traffic. Trio starts to saturate at $L = 50\%$, which is observable by the reduced gain. For a share $\geq 85\%$, there the load has a clear impact on the gain. It raises from $\approx 10\%$ at $L = 40\%$ to $\approx 16\%$ at $L = 50\%$. In summary, choosing the correct topology configuration becomes more important with higher traffic skew and load.

Figures 7.8b and 7.8c illustrate the impact of the flow size distribution of the skewed traffic on the gain. The reference configurations are Duo $(2, 0, 6)$ and Opera, respectively. First, for Duo (Figure 7.8b), the gain diminishes for all distributions when x increases. For higher shares, the optimal configuration has smaller k_{do} becomes more similar to $(2, 0, 6)$. In fact, for $x = 100\%$, $(2, 0, 6)$ is the best configuration.

Compared to Opera (Figure 7.8c), Trio shows improvements of up to 10% when skewed traffic is sampled from the Datamining or Hadoop distribution. The numbers vary depending on the exact share. For Websearch, the gain strongly grows with x.[12] As observed in Section 6.3 and also in prior work [49], Opera's throughput (and consequently the goodput) collapses if the amount of traffic routed via the expander part increases, as is the case with the Websearch distribution. As a general trend, we observe that the gains decrease with the average flow size of the distributions (Datamining has the highest (≈ 5.5 MB), followed by Hadoop (≈ 3.66 MB) and then Websearch (≈ 1.55 MB)). The intuition is that Hadoop and Websearch are per se less skewed than Datamining, which overall benefits Opera. In summary, the gains from Trio also persist for other distributions of skewed traffic.

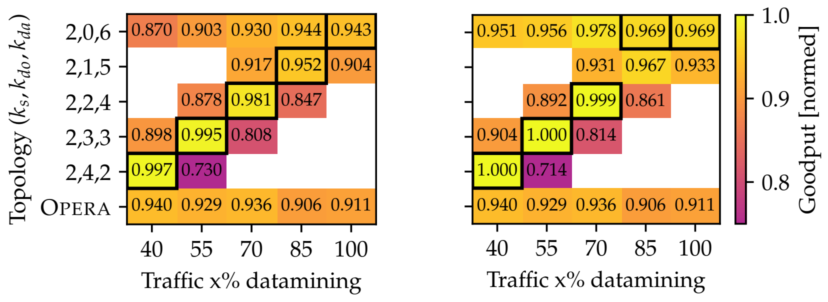
LocalLB preserves competitive goodput at lower complexity. In the previous analyses, Trio uses LLB as proposed in Section 7.4.1.3. This scheduling can run rack-local, which reduces the synchronization overhead compared to running Trio with the RLB scheduling [48, Algorithm 1]. To investigate how the two schedulers fare against each other, Figure 7.9 shows the goodput and runtime of both scheduling approaches. Note that the simulation runtime does not reflect

[12]The upper end of the figure is cut. The maximum gain is 112% at $x = 100\%$ Websearch.



(a) Matching traffic and topology configuration. **(b)** 70% DATAMINING.

Figure 7.9 Comparison of TRIO's scheduling (LocalLB) and RotorLB scheduling. Load 40%. The goodput is normalized to the offered traffic, the runtime is normalized to the LocalLB - (2,1,5). The errorbars indicate the 95%-confidence interval. LocalLB maintains similar performance as RotorLB but has lower runtimes that hint at lower complexity.

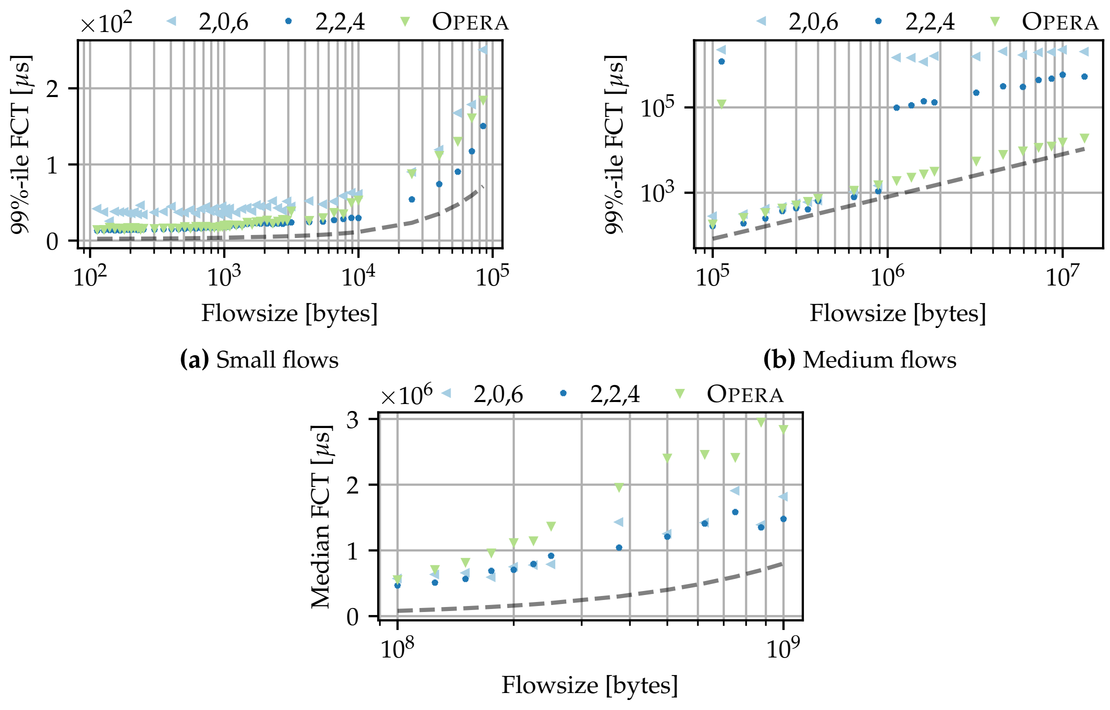


(a) r_{da} = 10ms **(b)** r_{da} = 100µs

Figure 7.10 Goodput averaged over 1s of simulation. Comparison of reconfiguration times of the DA links with fixed duty cycle. The heatmap compares topology configurations and traffic mixes. On a high-level, the reconfiguration delay does not impact the benefits of matching traffix mix and topology configuration. Inspecting the values individually, we cannot identify a consistent pattern.

the actual synchronization effort, which also depends on hardware characteristics, but can give a first indication. The figure compares two cases: (1) the topology matching the traffic share, i.e., (2,1,5) is run with $x = 85\%$, (2,2,4) with $x = 70\%$ etc., and (2) all topology configurations are run on $x = 70\%$. For all configurations, the differences in goodput between LLB and RLB are $< 10\%$. For the matching traffic, we observe that LLB achieves slightly higher goodput than RLB and, for the fixed share, the opposite is the case. LLB consistently has lower runtimes than RLB. The difference increases with the number of DOBL ports and emphasizes the gain of avoiding inter-ToR synchronization for indirect traffic on the DOBL links.

TRIO's gain persists for other reconfiguration delays Figure 7.10 illustrates similar heatmaps as in Figure 7.7 for r_{da} = 10ms (Figure 7.10a) and r_{da} = 100µs (Figure 7.10b). The reconfiguration period is adjusted so that the duty cycle remains constant. The load is $L = 40\%$. On a macroscopic level, the behavior is similar as before: matching the topology configuration to the traffic mix



(a) Small flows **(b)** Medium flows **(c)** Large flows

Figure 7.11 Comparison of flow completion times (FCTs) against the flow size. Flows are separated into three groups showing the 99%-ile for small flows (a) and medium flows (b), and the median for large flows (c). Load is 40%, traffic mix with 70% Datamining and 30% uniform. Trio achieves lower FCT than Opera for small and large flows. Results for medium-sized flows vary depending on the used topology part.

maximizes the achieved goodput. Comparing the values in detail, no consistent impact of the reduction of r_{da} (and the implied reduction of the reconfiguration period) is observable. Some combinations have higher, and some have lower achieved goodput.

7.4.2.3 Individual Flows' Performance

Besides the goodput, this section evaluates the impact on the individual flows. Therefore, it first evaluates the FCT of flows of different sizes. The second part investigates packet reordering.

Trio preserves competitive flow completion times. To this end, Figure 7.11 visualizes the FCTs against the flow sizes for Trio, Duo (2,0,6) and Opera. The flows are separated into three groups for the sake of readability. The dashed lines indicate the optimal transmission time accounting for queues and propagation.[13] For small flows (≤ 100 KB, Figure 7.11a), we consider the 99%-ile to better understand how this latency-sensitive traffic is handled. For all sizes, Trio achieves the smallest FCT. It is followed by Opera and Duo. Recall that small flows use only the static topology. Compared to Trio, Duo's performance suffers from traffic belonging to the uniform traffic that cannot be served well over the Da links and creates congestion on the Static links.

For medium-sized flows (99%-ile, Figure 7.11b), Trio consistently performs better than Duo for the same reason as for the small flows. Trio also outperforms Opera until Trio starts classifying flows for the Da part (1 MB). From thereon, flows are using TCP, and there is a strong increase in the FCT, an effect that has already been observed when evaluating Duo (Section 6.3) and similarly

[13]Using the calculations provided by [49].

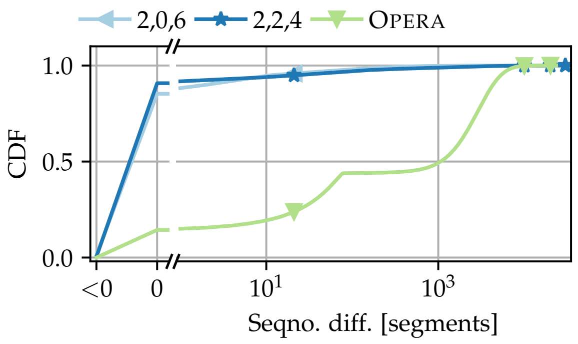


Figure 7.12 CDF of the difference expected and received sequence number as indicator for packet reordering. $L = 40\%$ load, $x = 70\%$ DATAMINING. TRIO has less packet reordering than DUO and OPERA.

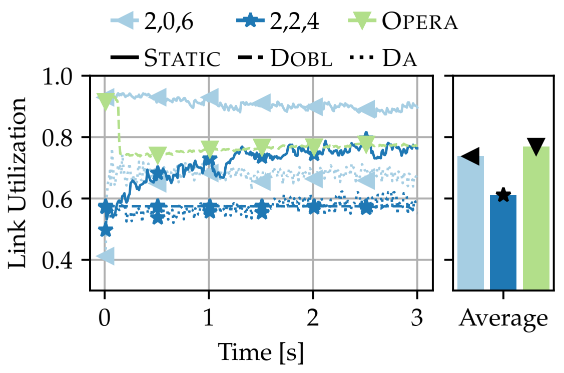


Figure 7.13 Link utilization separated by sub-topology over time and average. $L = 40\%$ and $x = 70\%$ DATAMINING share. Overall, matching the topology configuration with TRIO results in lower link utilizations.

also for OPERA [49] (for flows > 15 MB). A special point of interest are flows of size 112.5 kB, which arrive according to uniform matrices. All systems show an increased FCT here. OPERA is the best, followed by TRIO (with the optimized configuration) and DUO (2,0,6). This steep increase can be explained by the fact that these flows do not fit in a single DOBL slot and need multiple cycles of the DOBL sub-topology to be transmitted.

Lastly, the situation changes for the FCTs of large flows (≥ 100 MB, Figure 7.11c). We consider the median value here and observe that TRIO performs best, followed by DUO and OPERA. Specifically, the difference between TRIO and DUO is smaller than between TRIO and OPERA; in fact, the FCTs for DUO (2,0,6) and TRIO overlap for most flow sizes. This relates to our previous observations regarding the achieved goodput.

In summary, integrating DOBL into a purely demand-aware topology improves the FCT for all flow sizes. Compared to OPERA, TRIO trades off benefits for the small and large flows against those of medium-sized and the DOBL flows.

TRIO has a moderate amount of packet reordering. To take a different perspective on the impact on the individual flows, Figure 7.12 illustrates the difference between the expected and received sequence number of the packets at the receiver. Thereby, it hints at the packet reordering experienced in the different topologies. The abscissa is broken into values < 0 (useless re-transmissions), = 0 and more fine-grained for values > 0 (reception of reordered packets). Comparing the systems, we observe that the behavior of $(2, 0, 6)$ (DUO) and $(2, 2, 4)$ is similar. For both, > 80% of the packets arrive in order. This is fundamentally different for OPERA, for which < 20% of the packets arrive in order. The shape of the remaining curve has a break around 79 packets which corresponds to the number of packets needed to transmit the flows from the uniform traffic (of size 112.5 kB). We conclude that OPERA requires more effort from the receivers to forward packets in the right order to the applications.

7.4.2.4 Resource utilization

Figure 7.13 shows the link utilization over time and on average for a single run. For each topology, the present link types are shown separately (OPERA has only DOBL, DUO has only STATIC and DA,

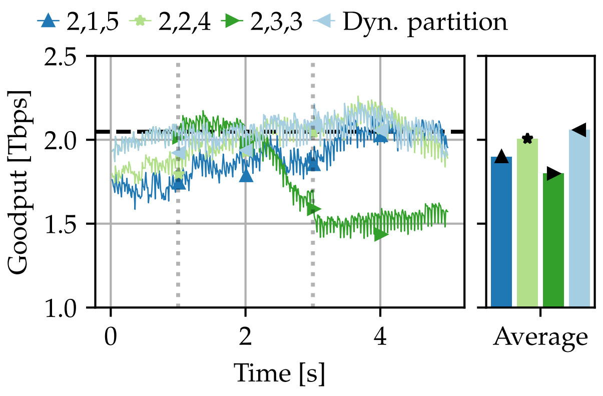


Figure 7.14 Goodput over time and average in scenario with varying traffic mix. The dotted vertical lines indicate changes in the traffic mix (at 1 s and 3 s). The dynamic partitioning that continuously matches the sizes of the sub-topologies to the traffic mix achieves the highest throughput.

and Trio has all three). After the initial transient phase, the utilizations are stable over time and vary only a little. For $(2, 0, 6)$, the utilization of Static links is around 90%, whereas Da links are utilized by 70%. Opera's Dobl links are ≈ 80% utilized. For $(2, 2, 4)$, Dobl and Da have a similar utilization of 60%, and Static links are utilized by 80%. We note that Trio has the lowest utilization on average but achieves the highest goodput, demonstrating the benefits of matching traffic and topology also in the dimension of resource efficiency.

7.4.2.5 Dynamic partitioning preserves gains for varying traffic mixes

As a last aspect, we assess the benefits of dynamic partitioning in Trio. Figure 7.14 shows the goodput over time and the average of a simulation in which the traffic mix varies over time. Specifically, x increases from 55% for $< 1\,\mathrm{s}$ to 70% ($1\,\mathrm{s} \leq t < 3\,\mathrm{s}$) to 85% ($\geq 3\,\mathrm{s}$). For each of the three phases, the figure shows the goodput of the matching configuration $(2, 3, 3)$, $(2, 2, 4)$ and $(2, 1, 5)$ respectively. For these three *static* baselines, we see in the period with matching traffic that they achieve the offered load (after some transient phase) but drop in goodput when not matching the traffic. For instance, $(2, 3, 3)$ significantly drops after 1 s. In contrast, for the dynamic partitioning, the achieved goodput aligns with the offered load throughout the run. The superiority of the dynamic partitioning also becomes evident from the average values as it achieves the highest value. Note that the shown example is hard-coded and only demonstrates the benefits of applying the dynamic partitioning. The design of an (online) optimization algorithm is left for future work.

7.4.3 Takeaways

This section presents Trio, an end-to-end design of an RDCN with macroscopic reconfiguration-awareness. Like Cerberus, it combines Static, Da, and Dobl sub-topologies into a unified system. Trio can match the topology configuration and dimensioning of Da, Dobl and Static sub-topologies over time to the present traffic. Thereby, it improves goodput compared to existing RDCN designs in various traffic conditions. Moreover, it mainly preserves the completion times of individual flows. Whereas this evaluation demonstrates the benefits of matching traffic and topology in static traffic conditions, it also shows that adapting the topology dimensioning during runtime is possible.

7.5 Summary

This chapter takes a first step towards topology reconfiguration-aware RDCNs concerning the reconfiguration class. Therefore, it presents two hybrid RDCNs that combine sub-topologies with *DA* and *DO* reconfigurations with a static base topology. The rational is that depending on the traffic, other topology reconfiguration mechanisms are beneficial. CERBERUS introduces the idea of matching the sub-topologies to the traffic. TRIO presents an end-to-end design of such a system that is evaluated using packet-level simulations. Besides demonstrating the viability of this approach, the evaluation also illustrates the advantages of the presented systems in terms of goodput – up to 15% improvement over purely *DA* or *DO* proposals. Moreover, it maintains and for some traffic even improves the flow completion times. With the introduction of *LocalLB*, TRIO also advances on a third front: it reduces the synchronization effort for the sub-topology. Finally, the network and transport layer design of TRIO provide means to dynamically adapt the dimensions of the sub-topologies at runtime, a crucial aspect for matching the traffic over time.

While all these features improve the performance, there are still some points for future work. This chapter focuses on the performance of the hybrid RDCNs as the most essential aspect. It mainly revolves around simulation results, a more in depth analytical modeling is provided in [3, 4] and another thesis. Also more detailed investigations with respect to failures and detailed cost models are left for future work.[14] However, such discussions become only relevant if a performance benefit can be identified in the first place. Lastly, the design of optimization algorithms for adapting the sizes of the sub-topologies over time is a natural avenue for future work.

[14]Since TRIO uses similar topology components as DUO, the failure discussion from Section 6.5 is also applicable here.

Chapter 8

Conclusion and Outlook

As the traffic demand in today's communication networks continues to grow, operators face new challenges for the efficient operation of their networks. Besides the growth in volume, new application types and more stringent requirements towards latency and reliability challenge network operators. In the past years, researchers have come up with the idea of topological reconfigurability at run-time to provide means to address these challenges. The innovation of optical switching technologies that can quickly change links has led to a wide gamut of different solutions for both wide area networks (WANs) and datacenter networks (DCNs).

However, while topological reconfigurations, in principle, can help to operate networks more efficiently, they also introduce new challenges: The link's capacity is temporarily unavailable, endpoints must establish new links, and network controllers must update their forwarding tables. Depending on the (type of) equipment in use, reconfiguration delays can vary between minutes, e.g., for reconfigurable optical add-drop multiplexers (ROADMs) in WANs down to nanoseconds for the fastest switching equipment in DCNs. Also, the link's endpoints add to the reconfiguration delay and, in particular here, measurement procedures and a deeper understanding of the components' behavior under reconfigurations are needed.

Topological reconfigurations must be optimized in order to obtain the benefits of adaptivity. This optimization can also include reconfigurations on other layers, such as the IP routing and the demand, to further increase the benefits. Specifically in WAN environments in which Internet Service Providers (ISPs) cooperate with content providers (CPs), such a joint optimization and reconfiguration of three layers adds new potential for efficiency to be explored.

Operation and implementation of reconfigurable topologies require control planes to become faster. In particular, in reconfigurable datacenter networks (RDCNs), where frequent adaptations of the topology are considered, obtaining a consistent (global) view of the network as required by centralized software defined networking (SDN) controllers is hard. For distributed network controllers, state propagation may become a bottleneck. As a result, dynamic optical links are often not fully integrated into the routing, i.e., flows are either routed on the static or on the dynamic topology part.

In the past, several classes of topological reconfigurability have emerged. demand-aware (*DA*) reconfigurations optimize the topology toward measurements or estimates of the demand, whereas demand-oblivious (*DO*) reconfigurations follow pre-calculated schedules. Both are most effective for different traffic patterns. Hence, choosing the right reconfiguration class for the present traffic can further improve the performance of the network. Overall, integrating awareness of the topological reconfigurations in the network is necessary to utilize their potential fully.

8.1 Summary

This thesis explores approaches adding macroscopic reconfiguration-awareness in *DA* reconfigurable networks to improve network efficiency. Therefore, it investigates the characteristics of reconfigurable network topologies along three characteristics: the reconfiguration delay, the considered networking layers for reconfigurations (topology, routing, and demand), and the considered classes of reconfigurations (*DA* and *DO*). The contributions of this thesis range from measurement procedures and campaigns over mathematical optimizations to new network designs that are evaluated using simulations and prototype implementations. This section summarizes the major points in the following:

Modeling and Evaluation of Topological Reconfigurability. The reconfiguration delay can be a significant factor for efficiency in reconfigurable topologies, in particular, when the topology is reconfigured frequently. While existing measurements of topological reconfigurations focus on singular settings and the reconfiguration time of the physical layer, this thesis takes a more holistic view of the end-to-end reconfiguration delay of a link. To this end, it evaluates the length of interruption on the data plane and the delay on the control plane for various reconfiguration scenarios and commercially off-the-shelf (COTS) data plane devices. The results demonstrate that there are indeed significant differences between the devices and scenarios.

Moreover, this thesis presents an experimentation framework for exploring the impacts of reconfiguration classes, namely *DA* and *DO* reconfigurations. The proposed solution ExReC relies only on COTS equipment and can be configured to emulate topologies from all *DA* to all *DO*.

Joint optimization and reconfiguration of topology and demand. Topological reconfigurations unavoidably lead to reconfigurations on higher networking layers as well. The question arises if and how the higher layers should be included in the optimization. Therefore, this thesis presents a multi-layer optimization problem considering joint optimization and reconfiguration of topology, routing, and demand layer. The evaluation using demand data from a large ISP demonstrates the benefits of such a joint optimization approach which culminates in up to 15% of deployed network capacity saved on various time scales. Moreover, we elaborate that such reconfigurations are feasible using today's technologies. They mainly lead to small links being modified while maintaining a stable core topology for most of the traffic.

Design of a High-throughput RDCN facilitating integrated multi-hop routing with local control. Frequent topological reconfigurations require the control plane to propagate network state updates quickly. As a result, many solutions fall back to segregated routing, which separates traffic on the static and on the dynamic sub-topologies. To alleviate this and to investigate the potential of integrated routing over both sub-topologies, this thesis presents Duo, a high throughput RDCN with local routing and control. Duo utilizes the structure of de Bruijn-graph as a static base topology to minimize the effort to update the nodes' forwarding tables: the updates can be calculated with local information. The evaluation using a packet-level simulator demonstrates the benefits of integrated multi-hop routing in terms of higher throughput and improved flow completion times. Moreover, it shows the feasibility of Duo by presenting a proof-of-concept implementation.

Design of Macroscopic Topology Reconfiguration-aware Networks for datacenters (DCs). The two reconfiguration classes, *DA* (considering the demand for adapting the topology) and *DO* (following a pre-calculated scheme for adapting the topology), differ in their performance depending on the demand in the network. This opens opportunities to increase the network performance further using the reconfiguration class that is best for the present traffic. Thereby, the topology becomes reconfiguration-aware on a macroscopic level. This thesis presents two solutions which consider three sub-topologies (static, *DA* reconfigurations, and *DO* reconfigurations) and assign traffic to the sub-topology that provides the best performance. We present an end-to-end design of such a hybrid, multi-reconfiguration-class RDCN and evaluates it using packet-level simulations. Moreover, this thesis empirically shows that matching the sizes of the sub-topologies to the share of the respective traffic is crucial for efficient network operation for throughput and flow completion times. As the traffic mix may change over time, it further demonstrates that an adaptation of the topology over time is beneficial and shows avenues for how to achieve this.

8.2 Future Work

This thesis presents the first steps towards reconfiguration-awareness in reconfigurable topologies. It demonstrates that already a macroscopic perspective, i.e., by differentiating classes of reconfigurations and the involved layers, increases the performance and efficiency of network operation. However, there are several avenues for future work:

This thesis focuses on the performance improvements due to macroscopic reconfiguration-awareness, e.g., in Chapter 5, 6, and 7, but leaves in-depth evaluations of failures and more detailed cost models like presented in [15] aside. While these additional evaluation aspects are critical per se, they are only relevant if the proposed systems improve performance, as demonstrated in this thesis. Thus, they are a intuitive extension for future work.

In addition, some design aspects allow more detailed investigations. For instance, the presented solutions Duo and Trio use de Bruijn-graph-based backbone topologies. De Bruijn graphs are only one alternative for network topologies that support local routing. The evaluation of other designs such as hypercube or Butterfly networks [304] or distributed hash tables such as Kademlia [306] might be beneficial. This also applies to the used transport protocols. While NDP and TCP are widely deployed and hence, natural choices, more specialized variants, e.g., ReTCP [204] or Time division TCP [205], might improve the performance even further.

Section 7.4 sketches the idea of adapting the reconfiguration-classes over time. It presents a system that can modify the sub-topology sizes by reassigning the switches' ports to different scheduler but misses an (online) optimization algorithm. While the packet-level simulation-based evaluation in Chapter 7 hints at the feasibility of the approach, a proof thereof using an end-to-end system implementation is still pending. Moreover, this idea can also be applied to the layers considered for reconfiguration. In principle, this can help to reduce the reconfiguration cost and the run-time of the algorithm. The latter cost might be particularly interesting to deploy the proposed joint optimization approach from Chapter 5. Currently, the algorithm run-time is not evaluated.

Another stream for future work is to investigate the feasibility and benefits of the microscopic-perspective. This thesis focuses on the macroscopic classification, but it is still an open question if a more fine granular optimization or adaptation of reconfiguration classes and the layers brings further performance improvements. Specifically, this means to formulate models, to design algorithms and to build systems that optimize reconfigurations on a link by link basis.

Lastly, with the advent of 6G requirements, more stringent requirements toward end-to-end performance are formulated. This thesis considers reconfigurations in two crucial components (or domains) of end-to-end connections, the WAN and DCN but separately. To this end, an extension towards a multi-domain approach that jointly considers the WAN and DCN component can be a next step. Specifically, answering the questions when to use reconfiguration on which component and when joint optimization and reconfiguration are feasible and beneficial might add a new dimension of topological reconfiguration-awareness.

List of Figures

List of Tables

Abbreviations

ALTO	Application-Layer Traffic Optimization	82
AWG	arrayed waveguide grating	139
BCT	batch completion time	74
BTS	Binary Tree Star	15
BvN decomposition	Birkhoff-von Neumann decomposition	41
BVT	bandwidth-variable transceiver	34
CAPEX	capital expenditures	77
CDF	cumulative distribution function	170
CDN	content delivery network	173
CP	content provider	165
COTS	commercially off-the-shelf	166
CPU	central processing unit	129
DA	demand-aware	169
DC	datacenter	167
DCN	datacenter network	173
DCT	demand completion time	135
DML	distributed machine learning	169
DNS	domain name system	82
DO	demand-oblivious	169
DPDK	Data Plane Development Kit	170
DWDM	Dense Wavelength Division Multiplexing	78
ECMP	equal cost multi-path	44
EON	elastic optical network	106
EPS	electronic packet switch	139
FCT	flow completion time	170
FPGA	Field Programmable Gate Array	49
GPU	graphics processing unit	44
IGP	Interior Gateway Protocol	82
ILP	Integer Linear Program	36
IMH	integrated multi-hop	171

INT	in-band network telemetry	58
ISP	Internet Service Provider	173
LC	liquid crystal	30
LLB	LocalLB	150
LP	Linear Program	37
LPM	longest prefix match	155
MEMS	microelectromechanical system	49
MIP	Mixed Integer Program	89
MILP	Mixed Integer Linear Program	36
ML	machine learning	105
MPI	Message Passing Interface	14
NN	Neural Network	42
NCCL	NVIDIA Collective Communication Library	14
NFV	network function virtualization	37
NIC	network interface card	109
OCS	optical circuit switch	139
OPEX	operational expenditures	88
PCE	Path Computation Element	82
PNI	private network interconnect	170
PoP	point of presence	170
PS	parameter server	43
QoS	quality of service	77
QoE	quality of experience	80
RDCN	reconfigurable datacenter network	169
RDMA	Remote Direct Memory Access	155
RLB	RotorLB	150
ROADM	reconfigurable optical add-drop multiplexer	165
SDN	software defined networking	165
SGD	Stochastic Gradient Descent	12
SNR	signal-to-noise ratio	36
SR	Segment Routing	82
S-SGD	Synchronous SGD	17
TCP	transport control protocol	108
TDMA	Time Division Multiple Access	48
TE	traffic engineering	84

TMT	ToR-Matching-ToR	170
ToR	top-of-rack	173
TPU	tensor processing unit	14
VLAN	Virtual LAN	58
VLB	Valiant load balancing	137
VM	virtual machine	67
VN	virtual network	37
VNE	virtual network embedding	37
WAN	wide area network	169
WDM	Wavelength Division Multiplexing	50
WSS	wavelength selective switch	49

Publications by the Author

Journal Articles

[1] Johannes Zerwas, Ingmar Poese, Stefan Schmid, and Andreas Blenk. “On the Benefits of Joint Optimization of Reconfigurable CDN-ISP Infrastructure”. In: *IEEE Transactions on Network and Service Management* 19.1 (Mar. 2022). © 2021 IEEE. Reprinted, with permission., pp. 158–173.

[2] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. “Duo: A high-throughput reconfigurable datacenter network using local routing and control”. In: *Proc. ACM Meas. Anal. Comput. Syst.* 7.1 (Mar. 2023), pp. 1–25.

[3] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. “Cerberus: The Power of Choices in Datacenter Topology Design (A Throughput Perspective)”. In: *Proc. ACM Meas. Anal. Comput. Syst.* 5.3 (Dec. 2021), pp. 1–33.

[4] Johannes Zerwas, Chen Griner, Stefan Schmid, and Chen Avin. “D3: A Dynamically Self-Adjusting Datacenter Network for Throughput Maximization”. In: *Under submission* (2024).

[5] Johannes Zerwas, Patrick Kalmbach, Stefan Schmid, and Andreas Blenk. “Ismael: Using Machine Learning To Predict Acceptance of Virtual Clusters in Data Centers”. In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 950–964.

[6] Péter Babarczi et al. “A mathematical framework for measuring network flexibility”. In: *Computer Communications* 164 (2020), pp. 13–24.

[7] Ferenc Mogyorosi, Peter Babarczi, Johannes Zerwas, Andreas Blenk, and Alija Pasic. “Resilient Control Plane Design for Virtualized 6G Core Networks”. In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 2453–2467.

[8] Patrick Krämer, Oliver Zeidler, Philip Diederich, Johannes Zerwas, Andreas Blenk, and Wolfgang Kellerer. “Mistill: Distilling Distributed Network Protocols from Examples”. In: *IEEE Transactions on Network and Service Management* (2023), pp. 1–1.

[9] Kaan Aykurt, Johannes Zerwas, Andreas Blenk, and Wolfgang Kellerer. “When TCP Meets Reconfigurations: A Comprehensive Measurement Study”. In: *IEEE Transactions on Network and Service Management* (2023), pp. 1–1.

Conference Proceedings

[10] Johannes Zerwas, Kaan Aykurt, Stefan Schmid, and Andreas Blenk. “Network Traffic Characteristics of Machine Learning Frameworks Under the Microscope”. In: *17th International Conference on Network and Service Management (CNSM 2021)*. Izmir, Turkey, Oct. 2021, pp. 207–215.

[11] Johannes Zerwas, Wolfgang Kellerer, and Andreas Blenk. "What You Need to Know About Optical Circuit Reconfigurations in Datacenter Networks". In: *the 33nd International Teletraffic Congress (ITC 33)*. Avignon, France, Aug. 2021, pp. 1–9.

[12] Johannes Zerwas, Chen Avin, Stefan Schmid, and Andreas Blenk. "ExReC: Experimental Framework for Reconfigurable Networks Based on Off-the-Shelf Hardware". In: *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. Layfette, IN, USA: ACM, Dec. 2021, pp. 66–72.

[13] Enric Pujol, Ingmar Poese, Johannes Zerwas, Georgios Smaragdakis, and Anja Feldmann. "Steering Hyper-Giants' Traffic at Scale". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. Orlando, Florida, Dec. 2019, pp. 82–95.

[14] Andreas Blenk, Patrick Kalmbach, Johannes Zerwas, Michael Jarschel, Stefan Schmid, and Wolfgang Kellerer. "NeuroViNE: A Neural Preprocessor for Your Virtual Network Embedding Algorithm". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Honolulu, HI, USA, Apr. 2018, pp. 405–413.

[15] Alberto Martínez Alba, Péter Babarczi, Andreas Blenk, Mu He, Patrick Krämer, Johannes Zerwas, and Wolfgang Kellerer. "Modeling the Cost of Flexibility in Communication Networks". In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. Virtual, 2021, pp. 1–10.

[16] Johannes Zerwas, Patrick Kalmbach, Carlo Fuerst, Arne Ludwig, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. "Ahab: Data-Driven Virtual Cluster Hunting". In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. Zurich, Switzerland, 2018, pp. 1–9.

[17] Patrick Kalmbach, Johannes Zerwas, Péter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. "Empowering Self-Driving Networks". In: *Proceedings of the Afternoon Workshop on Self-Driving Networks*. Budapest, Hungary, 2018, pp. 8–14.

[18] Johannes Zerwas, Patrick Kalmbach, Laurenz Henkel, Gabor Retvari, Wolfgang Kellerer, Andreas Blenk, and Stefan Schmid. "NetBOA: Self-Driving Network Benchmarking". In: *Proceedings of the 2019 Workshop on Network Meets AI & ML*. Beijing, China, 2019, pp. 8–14.

[19] Amaury Van Bemten, Nemanja Deric, Johannes Zerwas, Andreas Blenk, Stefan Schmid, and Wolfgang Kellerer. "Loko: Predictable Latency in Small Networks". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. Orlando, Florida, Dec. 2019, pp. 355–369.

[20] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. "Cerberus". In: *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*. Mumbai, India, June 2022, pp. 99–100.

[21] Kaan Aykurt, Johannes Zerwas, Andreas Blenk, and Wolfgang Kellerer. "On the Performance of TCP in Reconfigurable Data Center Networks". In: *2022 18th International Conference on Network and Service Management (CNSM)*. Thessaloniki, Greece, Oct. 2022, pp. 127–135.

[22] Johannes Zerwas, Patrick Krämer, Răzvan-Mihai Ursu, Navidreza Asadi, Phil Rodgers, Leon Wong, and Wolfgang Kellerer. “KAPETÁNIOS: Automated Kubernetes Adaptation through a Digital Twin”. In: *2022 13th International Conference on Network of the Future (NoF)*. Ghent, Belgium, Oct. 2022, pp. 1–3.

[23] Sebastian Schmidt, Johannes Zerwas, and Wolfgang Kellerer. “AdFAT: Adversarial Flow Arrival Time Generation for Demand-Oblivious Data Center Networks”. In: *2023 19th International Conference on Network and Service Management (CNSM)*. 2023, pp. 1–5.

[24] Kaan Aykurt, Răzvan-Mihai Ursu, Johannes Zerwas, Patrick Krämer, Navidreza Asadi, Leon Wong, and Wolfgang Kellerer. “HyPA: Hybrid Horizontal Pod Autoscaling with Automated Model Updates”. In: *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2023, pp. 8–14.

Book Chapters

[25] Massimo Tornatore et al. “Alert-Based Network Reconfiguration and Data Evacuation”. In: *Computer Communications and Networks*. Cham: Springer International Publishing, 2020, pp. 353–377. ISBN: 978-3-030-44685-7.

[26] Andreas Blenk, Patrick Krämer, Johannes Zerwas, and Stefan Schmid. “Designing Algorithms for Data-Driven Network Management and Control: State-of-the-Art and Challenges”. In: *Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning*. John Wiley & Sons, Ltd, 2021. Chap. 8. ISBN: 978-1-119-67550-1.

Technical Reports

[27] Johannes Zerwas, Kaan Aykurt, Stefan Schmid, and Andreas Blenk. *Dataset of “Network Traffic Characteristics of Machine Learning Frameworks Under the Microscope”*. PCAP. 2021. DOI: 10.14459/2021mp1632489.

Bibliography

[28] Harish Viswanathan and Preben E. Mogensen. “Communications in the 6G Era”. In: *IEEE Access* 8 (2020), pp. 57063–57074.

[29] Amin Shahraki, Mahmoud Abbasi, Md. Jalil Piran, and Amir Taherkordi. “A Comprehensive Survey on 6G Networks:Applications, Core Services, Enabling Technologies, and Future Challenges”. In: (Jan. 29, 2021). arXiv: 2101.12475 [cs.NI].

[30] Mostafa Zaman Chowdhury, Md. Shahjalal, Shakil Ahmed, and Yeong Min Jang. “6G Wireless Communication Systems: Applications, Requirements, Technologies, Challenges, and Research Directions”. In: *IEEE Open Journal of the Communications Society* 1 (2020), pp. 957–975.

[31] Alan Weissberger. *Telegeography: Global internet bandwidth rose by 28% in 2022*. IEEE ComSoc Technology Blog. Sept. 19, 2022. URL: https://techblog.comsoc.org/2022/09/19/telegeography-global-internet-bandwidth-rose-by-28-in-2022/ (visited on 05/30/2023).

[32] Anja Feldmann et al. “A year in lockdown”. In: *Communications of the ACM* 64.7 (July 2021), pp. 101–108.

[33] *Cisco Annual Internet Report (2018–2023) White Paper*. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 06/07/2023).

[34] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. “Inside the Social Network’s (Datacenter) Network”. In: *Proc. ACM SIGCOMM*. London, United Kingdom, Aug. 2015.

[35] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. “Understanding data center traffic characteristics”. In: *ACM SIGCOMM Computer Communication Review* 40.1 (Jan. 2010), pp. 92–99.

[36] Ashkan Aghdai, Fan Zhang, Nadun Dasanayake, Kang Xi, and H. Jonathan Chao. “Traffic measurement and analysis in an organic enterprise data center”. In: *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*. Taipei, Taiwan: IEEE, July 2013.

[37] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. “On the Complexity of Traffic Traces and Implications”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4.1 (May 2020), pp. 1–29.

[38] Jeremias Blendin, Fabrice Bendfeldt, Ingmar Poese, Boris Koldehofe, and Oliver Hohlfeld. “Dissecting Apple’s Meta-CDN during an iOS Update”. In: *Proc. ACM IMC*. Boston, MA, USA, 2018, pp. 408–414.

[39] Amedeo Sapio et al. “Scaling distributed machine learning with in-network aggregation”. In: (2019). arXiv: 1903.06701 [cs.NI].

[40] Nick McKeown et al. “OpenFlow”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), pp. 69–74.

[41] Pat Bosshart et al. “P4”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (July 2014), pp. 87–95.

[42] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. “A Scalable, Commodity Data Center Network Architecture”. In: *ACM SIGCOMM Computer Commun. Rev.* 38.4 (Oct. 2008), pp. 63–74.

[43] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. “Beyond Fat-Trees without Antennae, Mirrors, and Disco-Balls”. In: *Proc. ACM SIGCOMM '17*. Los Angeles, CA, USA, 2017, pp. 281–294.

[44] Arjun Singh et al. “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network”. In: *Proc. ACM SIGCOMM '15*. London, United Kingdom, 2015, pp. 183–197.

[45] Nathan Farrington et al. “Helios: a hybrid electrical/optical switch architecture for modular data centers”. In: *Proc. ACM SIGCOMM '10*. New Delhi, India, Aug. 2010, pp. 339–350.

[46] Kai Chen et al. “OSA: An optical switching architecture for data center networks with unprecedented flexibility”. In: *IEEE/ACM Trans. on Networking* 22.2 (2013), pp. 498–511.

[47] Monia Ghobadi et al. “Projector: Agile reconfigurable data center interconnect”. In: *Proc. ACM SIGCOMM '16*. Florianopolis, Brazil, 2016, pp. 216–229.

[48] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. “RotorNet. A Scalable, Low-complexity, Optical Datacenter Network”. In: *Proc. ACM SIGCOMM '17*. Los Angeles, CA, USA, Aug. 2017, pp. 267–280.

[49] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. “Expanding across time to deliver bandwidth efficiency and low latency”. In: *Proc. USENIX NSDI '20*. Santa Clara, CA, USA, Feb. 2020, pp. 1–18.

[50] He Liu et al. “Circuit switching under the radar with REACToR”. In: *Proc. USENIX NSDI '14*. Seattle, WA, USA, Apr. 2014, pp. 1–15.

[51] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and T. S. Eugene Ng. “A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-Tree”. In: *Proc. ACM SIGCOMM '17*. Los Angeles, CA, USA, Aug. 2017, pp. 295–308.

[52] George Porter et al. “Integrating microsecond circuit switching into the data center”. In: *ACM SIGCOMM Computer Commun. Rev.* 43.4 (2013), pp. 447–458.

[53] Hitesh Ballani et al. “Sirius: A Flat Datacenter Network with Nanosecond Optical Switching”. In: *Proc. ACM SIGCOMM '20*. New York, NY, USA (Virtual), Aug. 2020, pp. 782–797.

[54] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. “Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch”. In: *Proc. USENIX NSDI '17*. Boston, MA, USA, Mar. 2017, pp. 577–593.

[55] Kai Chen, Xitao Wen, Xingyu Ma, Yan Chen, Yong Xia, Chengchen Hu, and Qunfeng Dong. “WaveCube: A Scalable, Fault-Tolerant, High-Performance Optical Data Center Architecture”. In: *Proc. IEEE INFOCOM*. Kowloon, Hong Kong, Apr. 2015, pp. 1903–1911.

[56] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, T S Eugene Ng, Michael Kozuch, and Michael Ryan. “C-Through: Part-Time Optics in Data Centers”. In: *ACM SIGCOMM '10*. New Delhi, India, Aug. 2010, pp. 1–12.

[57] Gagan Choudhury, David Lynch, Gaurav Thakur, and Simon Tse. “Two Use Cases of Machine Learning for SDN-Enabled IP/Optical Networks: Traffic Matrix Prediction and Optical Path Performance Prediction [Invited]”. en. In: *J. Opt. Commun. Netw.* 10.10 (Oct. 2018), p. D52.

[58] Simon Tse and Gagan Choudhury. “Real-Time Traffic Management in AT&T's SDN-Enabled Core IP/Optical Network”. In: *Optical Fiber Communication Conference*. San Diego, California: OSA, Mar. 2018, Tu3H.2.

[59] Joshua L. Benjamin, Thomas Gerard, Domaniç Lavery, Polina Bayvel, and Georgios Zervas. “PULSE: Optical Circuit Switched Data Center Architecture Operating at Nanosecond Timescales”. In: *J. Lightwave Technol.* 38.18 (Sept. 2020), pp. 4906–4921.

[60] Ori Gerstel, Clarence Filsfils, Thomas Telkamp, Matthias Gunkel, Martin Horneffer, Victor Lopez, and Arturo Mayoral. “Multi-Layer Capacity Planning for IP-Optical Networks”. en. In: *IEEE Commun. Mag.* 52.1 (Jan. 2014), pp. 44–51.

[61] B. Ramamurthy and A. Ramakrishnan. “Virtual Topology Reconfiguration of Wavelength-Routed Optical WDM Networks”. In: *IEEE Globecom '00. Conference Record (Cat. No.00CH37137)*. Vol. 2. San Francisco, CA, USA, 2000, pp. 1269–1275.

[62] Ciril Rozic, Marco Savi, Chris Matrakidis, Dimitrios Klonidis, and Domenico Siracusa. “A Dynamic Multi-Layer Resource Allocation and Optimization Framework in Application-Centric Networks”. en. In: *J. Lightwave Technol.* 36.20 (Oct. 2018), pp. 4908–4914.

[63] Ioannis Tomkos et al. “Application Aware Multilayer Control and Optimization of Elastic WDM Switched Optical Networks”. en. In: *Optical Fiber Communication Conference*. San Diego, California: OSA, Mar. 2018, Th1D.4.

[64] Leon Poutievski et al. “Jupiter evolving”. In: *Proc. ACM SIGCOMM '22*. Amsterdam, NL, Aug. 2022.

[65] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H.Y. Wong, and Hongyi Zeng. “Robotron”. In: *Proc. ACM SIGCOMM '16*. Florianopolis, Brazil, Aug. 2016, pp. 426–439.

[66] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C. Mogul, and Amin Vahdat. *Gemini: Practical Reconfigurable Datacenter Networks with Topology and Traffic Engineering*. Oct. 15, 2021. arXiv: 2110.08374 [cs.NI].

[67] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. “OpenML: Networked Science in Machine Learning”. In: *ACM SIGKDD Explor. Newsl.* 15.2 (June 2014), pp. 49–60. ISSN: 1931-0145.

[68] Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. “PMLB: a large benchmark suite for machine learning evaluation and comparison”. In: *BioData mining* 10.1 (2017), pp. 1–13.

[69] Louis Columbus. *What's New In Gartner's Hype Cycle For AI, 2020*. Forbes. URL: https://www.forbes.com/sites/louiscolumbus/2020/10/04/whats-new-in-gartners-hype-cycle-for-ai-2020/ (visited on 06/10/2021).

[70] S. Lee, H. Kim, J. Park, J. Jang, C. Jeong, and S. Yoon. "TensorLightning: A Traffic-Efficient Distributed Deep Learning on Commodity Spark Clusters". In: *IEEE Access* 6 (2018), pp. 27671–27680.

[71] Yanping Huang et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism". In: *Advances in NIPS* 32 (2019), pp. 103–112.

[72] Zhen Zhang, Chaokun Chang, Haibin Lin, Yida Wang, Raman Arora, and Xin Jin. "Is Network the Bottleneck of Distributed Training?" In: *Proc. ACM Workshop on Network Meets AI & ML*. New York City, NY, USA, Aug. 2020, pp. 8–13.

[73] Alexander Sergeev and Mike Del Balso. *Horovod: fast and easy distributed deep learning in TensorFlow*. Feb. 15, 2018. arXiv: 1802.05799 [cs.LG].

[74] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. "KungFu: Making Training in Distributed Machine Learning Adaptive". In: *Proc. USENIX OSDI 20*. Virtual, Nov. 2020, pp. 937–954.

[75] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2016. arXiv: 1603.04467 [cs.LG]. URL: https://www.tensorflow.org/.

[76] *Apache SINGA*. URL: https://singa.apache.org/.

[77] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in NIPS 32*. 2019, pp. 8024–8035.

[78] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. 2020. arXiv: 1910.02054 [cs.LG].

[79] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536.

[80] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Communications of the ACM* 51.1 (Jan. 2008), pp. 107–113.

[81] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. "Parallelized Stochastic Gradient Descent". In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010.

[82] Mu Li et al. "Scaling Distributed Machine Learning with the Parameter Server". In: *Proc. USENIX OSDI '14*. Broomfield, CO, USA, Oct. 2014, pp. 583–598.

[83] Deepak Narayanan et al. "PipeDream: Generalized Pipeline Parallelism for DNN Training". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville, Ontario, Canada, Oct. 2019, pp. 1–15.

[84] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. "A survey on distributed machine learning". In: *ACM Computing Surveys (CSUR)* 53.2 (2020), pp. 1–33.

[85] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. “A Reliable Effective Terascale Linear Learning System”. In: *Journal of Machine Learning Research* 15.32 (2014), pp. 1111–1133.

[86] A. A. Awan, J. Bédorf, C. Chu, H. Subramoni, and D. K. Panda. “Scalable Distributed DNN Training using TensorFlow and CUDA-Aware MPI: Characterization, Designs, and Performance Evaluation”. In: *Proc. 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Larnaca, Cyprus, May 2019, pp. 498–507.

[87] *Google's Remote Procedure Call Library*. URL: http://www.grpc.io/.

[88] Edgar Gabriel et al. “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation”. In: *Proc. 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, Sept. 2004, pp. 97–104.

[89] *Gloo*. URL: https://github.com/facebookincubator/gloo.

[90] *NVIDIA - NCCL*. URL: https://github.com/NVIDIA/nccl.

[91] François Chollet et al. *Keras*. https://keras.io. 2015.

[92] T. Chen et al. “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems”. In: (2015). arXiv: 1512.01274 [cs.LG].

[93] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. “Asynchronous decentralized parallel stochastic gradient descent”. In: *Proc. ICML '18*. Stockholm, Sweden, July 2018, pp. 3043–3052.

[94] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. “Analysis of large-scale multi-tenant GPU clusters for DNN training workloads”. In: *Proc. USENIX ATC '19*. Renton, WA, USA, July 2019, pp. 947–960.

[95] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. “Deep gradient compression: Reducing the communication bandwidth for distributed training”. In: (2017). arXiv: 1712.01887 [cs.DC].

[96] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Andrew Y. Ng, and Bryan Catanzaro. “Deep Learning with COTS HPC Systems”. In: *Proc. ICML '13*. Atlanta, GA, USA, June 2013, pp. III-1337-III–1345.

[97] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H Campbell. “Tictac: Accelerating distributed deep learning with communication scheduling”. In: (2018). arXiv: 1803.03288 [cs.LG].

[98] Minsik Cho, Ulrich Finkler, Mauricio Serrano, David Kung, and Hillery Hunter. “BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy”. In: *IBM Journal of Research and Development* (Oct. 2019), pp. 1–1.

[99] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. “ATP: In-network Aggregation for Multi-tenant Learning”. In: *Proc. USENIX NSDI '21*. Virtual, Apr. 2021, pp. 741–761.

[100] Nadeen Gebara, Manya Ghobadi, and Costa Paolo. “In-network Aggregation for Shared Machine Learning Clusters”. In: *Proceedings of Machine Learning and Systems 3*. Virtual, Apr. 2021.

[101] Wenxin Li, Sheng Chen, Keqiu Li, Heng Qi, Renhai Xu, and Song Zhang. "Efficient Online Scheduling for Coflow-Aware Machine Learning Clusters". In: *IEEE Trans. on Cloud Computing* 10.4 (2022), pp. 2564–2579.

[102] Yunfeng Lu, Huaxi Gu, Xiaoshan Yu, and Peng Li. "X-NEST: A Scalable, Flexible, and High-Performance Network Architecture for Distributed Machine Learning". In: *Journal of Lightwave Technology* 39.13 (July 2021), pp. 4247–4254.

[103] Ling Liu, Qixuan Jin, Dan Wang, Hongfang Yu, Gang Sun, and Shouxi Luo. "PSNet: Reconfigurable network topology design for accelerating parameter server architecture based distributed machine learning". In: *Future Generation Computer Systems* 106 (2020), pp. 320–332.

[104] Mehrdad Khani et al. "TeraRack: A Tbps Rack for Machine Learning Training". In: 2020.

[105] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. "A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters". In: *Proc. USENIX OSDI '20*. Virtual, Nov. 2020, pp. 463–479.

[106] Tal Ben-Nun and Torsten Hoefler. "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis". In: *ACM Comput. Surv.* 52.4 (Aug. 2019).

[107] Shuai Wang, Dan Li, Jinkun Geng, Yue Gu, and Yang Cheng. "Impact of Network Topology on the Performance of DML: Theoretical Analysis and Practical Factors". In: *Proc. IEEE INFOCOM '19*. Paris, France, Apr. 2019, pp. 1729–1737.

[108] Shuqi Li, Yang Qin, Zukai Jiang, and Weihong Yang. "Efficient Communication Scheduling for Parameter Synchronization of DML in Data Center Networks". In: *IEEE Transactions on Network Science and Engineering* 9.4 (July 2022), pp. 1970–1985.

[109] Gyeongsik Yang, Changyong Shin, Jeunghwan Lee, Yeonho Yoo, and Chuck Yoo. "Prediction of the Resource Consumption of Distributed Deep Learning Systems". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.2 (May 2022), pp. 1–25.

[110] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA, June 2018, pp. 4510–4520.

[111] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: IEEE, July 2017, pp. 4700–4708.

[112] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778.

[113] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[114] Diederik Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[115] *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. Vol. 4. COURSERA: Neural networks for machine learning 2. 2012, pp. 26–31.

[116] A. Kumar. "Comparative performance analysis of versions of TCP in a local network with a lossy link". In: *IEEE/ACM Transactions on Networking* 6.4 (1998), pp. 485–498.

[117] Chen Avin and Stefan Schmid. "Toward demand-aware networking: A Theory for Self-Adjusting Networks". In: *ACM SIGCOMM Computer Communication Review* 48.5 (Jan. 2019), pp. 31–40.

[118] Klaus-Tycho Foerster and Stefan Schmid. "Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity". In: *ACM SIGACT News* 50.2 (2019), pp. 62–79.

[119] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. "A Survey of Reconfigurable Optical Networks". In: *Optical Switching and Networking* 41 (2021), p. 100621. ISSN: 1573-4277.

[120] William Maxwell Mellette, Glenn M. Schuster, George Porter, George Papen, and Joseph E. Ford. "A Scalable, Partially Configurable Optical Switch for Data Center Networks". In: *Journal of Lightwave Technology* 35.2 (Jan. 2017), pp. 136–144.

[121] S. Lange et al. "Sub-nanosecond Optical Switching Using Chip-Based Soliton Microcombs". In: *Optical Fiber Communication Conference (OFC) 2020*. San Diego, CA, USA: Optica Publishing Group, Mar. 2020.

[122] *Calient Optical Switches*. URL: https://www.calient.net/.

[123] R. Ryf et al. "1296-port MEMS Transparent Optical Crossconnect with 2.07Petabit/s Switch Capacity". In: *Optical Fiber Communication Conference and International Conference on Quantum Information*. Rochester, NY, USA: OSA, June 2001.

[124] Ryohei Urata et al. *Mission Apollo: Landing Optical Circuit Switching at Datacenter Scale*. Aug. 22, 2022. arXiv: 2208.10041 [cs.NI].

[125] Haoshuo Chen, Nicolas K. Fontaine, Roland Ryf, and David T. Neilson. "LCoS-Based Photonic Crossconnect". In: *Optical Fiber Communication Conference (OFC) 2019*. San Diego, CA, USA: OSA, Mar. 2019.

[126] Ciena Corporation. *What is ROADM?* Jan. 23, 2023. URL: https://www.ciena.com/insights/what-is/what-Is-roadm.html (visited on 01/23/2023).

[127] Reinhard Diestel. *Graph Theory*. Springer Berlin Heidelberg, 2017.

[128] Shay Vargaftik, Katherine Barabash, Yaniv Ben-Itzhak, Ofer Biran, Isaac Keslassy, Dean Lorenz, and Ariel Orda. "Composite-Path Switching". In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. Irvine, CA, USA: ACM, Dec. 2016.

[129] Ward Van Heddeghem, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. "A Quantitative Survey of the Power Saving Potential in IP-Over-WDM Backbone Networks". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 706–731.

[130] Filip Idzikowski, Luca Chiaraviglio, Antonio Cianfrani, Jorge Lopez Vizcaino, Marco Polverini, and Yabin Ye. "A Survey on Energy-Aware Design and Operation of Core Networks". In: *IEEE Communications Surveys & Tutorials* 18.2 (2016), pp. 1453–1499.

[131] Matthew Nance Hall, Paul Barford, Klaus-Tycho Foerster, Manya Ghobadi, William Jensen, and Ramakrishnan Durairajan. "Are WANs Ready for Optical Topology Programming?" In: *Proceedings of the ACM SIGCOMM 2021 Workshop on Optical Systems*. Virtual: ACM, Aug. 2021.

[132] Steven Gringeri, Nabil Bitar, and Tiejun J. Xia. "Extending software defined network principles to include optical transport". In: *IEEE Communications Magazine* 51.3 (Mar. 2013), pp. 32–40.

[133] A. Gencata and B. Mukherjee. "Virtual-Topology Adaptation for WDM Mesh Networks under Dynamic Traffic". en. In: *IEEE/ACM Trans. Networking* 11.2 (Apr. 2003), pp. 236–247.

[134] Ilker Akgun and Feza Buzluca. "Virtual Topology Reconfiguration on Optical WDM Networks Considering Traffic Grooming". en. In: *Optical Switching and Networking* 3.1 (July 2006), pp. 11–23.

[135] Víctor López, José Alberto Hernández, Óscar González de Dios, Juan Fernández Palacios, and Javier Aracil. "Multilayer Traffic Engineering for IP Over WDM Networks Based on Bayesian Decision Theory". In: *Journal of Optical Communications and Networking* 2.8 (July 2010), p. 515.

[136] Fernando Morales, Marc Ruiz, Lluís Gifre, Luis M. Contreras, Víctor López, and Luis Velasco. "Virtual Network Topology Adaptability Based on Data Analytics for Traffic Prediction". en. In: *J. Opt. Commun. Netw.* 9.1 (Jan. 2017), A35.

[137] Filip Idzikowski, Sebastian Orlowski, Christian Raack, Hagen Woesner, and Adam Wolisz. "Dynamic Routing at Different Layers in IP-over-WDM Networks — Maximizing Energy Savings". en. In: *Optical Switching and Networking* 8.3 (July 2011), pp. 181–200.

[138] Ward Van Heddeghem, Filip Idzikowski, Willem Vereecken, Didier Colle, Mario Pickavet, and Piet Demeester. "Power Consumption Modeling in Optical Multilayer Networks". In: *Photonic Network Communications* 24.2 (Jan. 2012), pp. 86–102.

[139] Panagiotis Melidis, Petros Nicopolitidis, and Georgios Papadimitriou. "Dynamic Threshold Reconfiguration Mechanisms for Green IP-Over-WDM Networks". en. In: *J. Lightwave Technol.* 34.18 (Sept. 2016), pp. 4354–4363.

[140] Chankyun Lee and June-Koo Kevin Rhee. "Traffic Grooming for IP-Over-WDM Networks: Energy and Delay Perspectives". en. In: *J. Opt. Commun. Netw.* 6.2 (Feb. 2014), p. 96.

[141] Thomas Fenz, Klaus-Tycho Foerster, and Stefan Schmid. "On Efficient Oblivious Wavelength Assignments for Programmable Wide-Area Topologies". In: *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. Layfette, IN, USA: ACM, Dec. 2021.

[142] S. Sinha and C.S.R. Murthy. "Information Theoretic Approach to Traffic Adaptive WDM Networks". en. In: *IEEE/ACM Trans. Networking* 13.4 (Aug. 2005), pp. 881–894.

[143] Andrea Bianco, Jorge Finochietto, and Chiara Piglione. "Cost and performance trade-offs in reconfiguration strategies for WDM networks". In: *2008 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks*. Venezia, Italy: IEEE, Feb. 2008, pp. 95–100.

[144] Phuong Nga Tran and Ulrich Killat. “Dynamic Reconfiguration of Logical Topology for WDM Networks under Traffic Changes”. en. In: *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*. Salvador, Bahia, Brazil: IEEE, Apr. 2008, p. 8.

[145] Edoardo Bonetto, Luca Chiaraviglio, Filip Idzikowski, and Esther Le Rouzic. “Algorithms for the Multi-Period Power-Aware Logical Topology Design With Reconfiguration Costs”. en. In: *J. Opt. Commun. Netw.* 5.5 (May 2013), p. 394.

[146] Y. Ohsita, T. Miyamura, S. Arakawa, S. Ata, E. Oki, K. Shiomoto, and M. Murata. “Gradually Reconfiguring Virtual Network Topologies Based on Estimated Traffic Matrices”. en. In: *IEEE/ACM Trans. Networking* 18.1 (Feb. 2010), pp. 177–189.

[147] Mohit Chamania, Marcel Caria, and Admela Jukan. “Achieving IP Routing Stability with Optical Bypass”. en. In: *2009 IEEE 3rd International Symposium on Advanced Networks and Telecommunication Systems (ANTS)*. New Delhi, India, Dec. 2009, pp. 1–3.

[148] Mohit Chamania and Admela Jukan. “A Comparative Analysis of the Effects of Dynamic Optical Circuit Provisioning on IP Routing”. en. In: *IEEE/ACM Trans. Networking* 22.2 (Apr. 2014), pp. 429–442.

[149] Bijoy Chand Chatterjee, Nityananda Sarma, and Eiji Oki. “Routing and Spectrum Allocation in Elastic Optical Networks: A Tutorial”. en. In: *IEEE Commun. Surv. Tutorials* 17.3 (2015), pp. 1776–1800.

[150] Takafumi Tanaka, Tetsuro Inui, Akihiro Kadohata, Wataru Imajuku, and Akira Hirano. “Multiperiod IP-Over-Elastic Network Reconfiguration With Adaptive Bandwidth Resizing and Modulation”. en. In: *J. Opt. Commun. Netw.* 8.7 (July 2016), A180.

[151] Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. “RADWAN”. In: *Proc. ACM SIGCOMM '18*. Budapest, Hungary: ACM, Aug. 2018.

[152] Rachee Singh, Nikolaj Bjorner, Sharon Shoham, Yawei Yin, John Arnold, and Jamie Gaudette. “Cost-effective capacity provisioning in wide area networks with Shoofly”. In: *Proc. ACM SIGCOMM '21*. Virtual: ACM, Aug. 2021.

[153] Su Jia, Xin Jin, Golnaz Ghasemiesfeh, Jiaxin Ding, and Jie Gao. “Competitive analysis for online scheduling in software-defined optical WAN”. In: *Proc. IEEE INFOCOM '17*. Atlanta, GA, USA: IEEE, May 2017, pp. 1–9.

[154] Michael Dinitz and Benjamin Moseley. “Scheduling for Weighted Flow and Completion Times in Reconfigurable Networks”. In: *Proc. IEEE INFOCOM '20*. Toronto, ON, Canada: IEEE, July 2020, pp. 1043–1052.

[155] Zhizhen Zhong, Nan Hua, Massimo Tornatore, Jialong Li, Yanhe Li, Xiaoping Zheng, and Biswanath Mukherjee. “Provisioning Short-Term Traffic Fluctuations in Elastic Optical Networks”. In: *IEEE/ACM Transactions on Networking* 27.4 (Aug. 2019), pp. 1460–1473.

[156] Shih-Hao Tseng. “Perseverance-Aware Traffic Engineering in Rate-Adaptive Networks with Reconfiguration Delay”. In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. Chicago, Illinois, USA: IEEE, Oct. 2019, pp. 1–10.

[157] Danilo Bórquez-Paredes, Alejandra Beghelli, and Ariel Leiva. “Network Virtualization Over Elastic Optical Networks: A Survey of Allocation Algorithms”. en. In: *Optical Fiber and Wireless Communications*. Ed. by Rastislav Roka. InTech, June 2017.

[158] Raouf Boutaba, Nashid Shahriar, and Siavash Fathi. "Elastic Optical Networking for 5G Transport". en. In: *Journal of Network and Systems Management* 25.4 (Oct. 2017), pp. 819–847.

[159] Sunny Shakya, Nabina Pradhan, Xiaojun Cao, Zilong Ye, and Chunming Qiao. "Virtual Network Embedding and Reconfiguration in Elastic Optical Networks". en. In: *2014 IEEE Global Communications Conference*. Austin, TX, USA: IEEE, Dec. 2014, pp. 2160–2165.

[160] Leonard Nonde, Taisir E. H. El-Gorashi, and Jaafar M. H. Elmirghani. "Energy Efficient Virtual Network Embedding for Cloud Networks". en. In: *J. Lightwave Technol.* 33.9 (May 2015), pp. 1828–1849.

[161] Jiawei Zhang, Yuefeng Ji, Mei Song, Hui Li, Rentao Gu, Yongli Zhao, and Jie Zhang. "Dynamic Virtual Network Embedding Over Multilayer Optical Networks". en. In: *J. Opt. Commun. Netw.* 7.9 (Sept. 2015), p. 918.

[162] Shihabur Rahman Chowdhury, Sara Ayoubi, Reaz Ahmed, Nashid Shahriar, Raouf Boutaba, Jeebak Mitra, and Liu Liu. "Multi-Layer Virtual Network Embedding". en. In: *IEEE Trans. Netw. Serv. Manage.* 15.3 (Sept. 2018), pp. 1132–1145.

[163] Francisco Carpio, Samia Dhahri, and Admela Jukan. "VNF Placement with Replication for Load Balancing in NFV Networks". en. In: *Proc. IEEE ICC 2017*. Paris, France: IEEE, May 2017, pp. 1–6.

[164] Andreas Blenk and Wolfgang Kellerer. "Traffic pattern based virtual network embedding". In: *Proc. ACM CoNEXT Student Workshop 2013*. Santa Barbara, California, USA: ACM, Dec. 2013, pp. 23–26.

[165] Fahad A. Ghonaim, Thomas E. Darcie, and Sudhakar Ganti. "Impact of SDN on Optical Router Bypass". In: *Journal of Optical Communications and Networking* 10.4 (Mar. 2018), pp. 332–343.

[166] Mirosl Aw Klinkowski and Krzysztof Walkowiak. "On the Advantages of Elastic Optical Networks for Provisioning of Cloud Computing Traffic". en. In: *IEEE Network* 27.6 (Nov. 2013), pp. 44–51.

[167] Jordi Perelló, Krzysztof Walkowiak, Mirosław Klinkowski, Salvatore Spadaro, and Davide Careglio. "Joint Content Placement and Lightpath Routing and Spectrum Assignment in CDNs over Elastic Optical Network Scenarios". en. In: *Computer Communications* 77 (Mar. 2016), pp. 72–84.

[168] Victor Lopez, Dimitrios Konidis, Domenico Siracusa, Ciril Rozic, Ioannis Tomkos, and Juan Pedro Fernandez-Palacios. "On the Benefits of Multilayer Optimization and Application Awareness". en. In: *J. Lightwave Technol.* 35.6 (Mar. 2017), pp. 1274–1279.

[169] Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu. "DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks". en. In: *Proceedings of the International Symposium on Quality of Service - IWQoS '19*. Phoenix, Arizona, USA: ACM, June 2019, pp. 1–10.

[170] Christoforos Kachris and Ioannis Tomkos. "A Survey on Optical Interconnects for Data Centers". In: *IEEE Communications Surveys & Tutorials* 14.4 (2012), pp. 1021–1036.

[171] Leslie G. Valiant. "A Scheme for Fast Parallel Communication". In: *SIAM J. Comput.* 11.2 (1982), pp. 350–361.

[172] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. "Re-architecting datacenter networks and stacks for low latency and high performance". In: *Proc. ACM SIGCOMM '17*. Los Angeles, CA, USA: ACM, Aug. 2017.

[173] Vamsi Addanki, Chen Avin, and Stefan Schmid. "Mars: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7.1 (Feb. 2023), pp. 1–43.

[174] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. "Proteus: A Topology Malleable Data Center Network". In: *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*. Monterey, California, USA, Oct. 2010, pp. 1–6.

[175] Min Yee Teh, Zhenguo Wu, and Keren Bergman. "Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering". In: *Journal of Optical Communications and Networking* 12.4 (Feb. 2020), B44.

[176] Xia Zhou et al. "Mirror mirror on the ceiling". In: *Proc. ACM SIGCOMM '12*. Helsinki, Finland: ACM, Aug. 2012, pp. 443–454.

[177] Navid Hamedazimi et al. "FireFly: A Reconfigurable Wireless Data Center Fabric Using Free-Space Optics". In: *Proc. ACM SIGCOMM '14*. Chicago, IL, USA: ACM, Aug. 2014, pp. 319–330.

[178] Janardhan Kulkarni, Stefan Schmid, and Paweł Schmidt. "Scheduling Opportunistic Links in Two-Tiered Reconfigurable Datacenters". In: *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. Virtual: ACM, July 2021.

[179] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. "SplayNet: Towards Locally Self-Adjusting Networks". In: *IEEE/ACM Transactions on Networking* 24.3 (June 2016), pp. 1421–1433.

[180] Chen Avin, Kaushik Mondal, and Stefan Schmid. "Push-Down Trees: Optimal Self-Adjusting Complete Trees". In: *IEEE/ACM Transactions on Networking* 30.6 (Dec. 2022), pp. 2419–2432.

[181] Evgeniy Feder, Ichha Rathod, Punit Shyamsukha, Robert Sama, Vitaly Aksenov, Iosif Salem, and Stefan Schmid. "Lazy Self-Adjusting Bounded-Degree Networks for the Matching Model". In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. London, United Kingdom: IEEE, May 2022, pp. 1089–1098.

[182] Shay Vargaftik, Cosmin Caba, Liran Schour, and Yaniv Ben-Itzhak. "C-Share: Optical Circuits Sharing for Software-Defined Data-Centers". In: *ACM SIGCOMM Comput. Commun. Rev.* 50.1 (Mar. 2020), pp. 2–9.

[183] Weitao Wang, Dingming Wu, Sushovan Das, Afsaneh Rahbar, Ang Chen, and T. S. Eugene Ng. "RDC: Energy-Efficient Data Center Network Congestion Relief with Topological Reconfigurability at the Edge". In: *Proc. USENIX NSDI '22*. Renton, WA, USA: USENIX Association, Apr. 2022, pp. 1267–1288.

[184] Xuwei Xue et al. "ROTOS: A Reconfigurable and Cost-Effective Architecture for High-Performance Optical Data Center Networks". In: *Journal of Lightwave Technology* 38.13 (July 2020), pp. 3485–3494.

[185] Jinzhe Che, Bingli Guo, Zichong Pang, Xuwei Xue, Yisong Zhao, Yuanzhi Guo, and Shanguo Huang. "RGAIA: a reconfigurable AWGR based optical data center network". In: *Optics Express* 30.13 (June 2022), pp. 23640–23655.

[186] Zuoqing Zhao et al. "ReSAW: a reconfigurable and picosecond-synchronized optical data center network based on an AWGR and the WR protocol". In: *Journal of Optical Communications and Networking* 14.9 (Aug. 2022), pp. 702–712.

[187] He Liu et al. "Scheduling techniques for hybrid circuit/packet networks". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. Heidelberg, Germany: ACM, Dec. 2015, pp. 1–13.

[188] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. "Costly Circuits, Submodular Schedules and Approximate Carathéodory Theorems". In: *ACM SIGMETRICS Performance Evaluation Review* 44.1 (June 2016), pp. 75–88.

[189] Xin Sunny Huang, Xiaoye Steven Sun, and T.S. Eugene Ng. "Sunflow: Efficient Optical Circuit Scheduling for Coflows". In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. Irvine, CA, USA: ACM, Dec. 2016, pp. 297–311.

[190] Klaus-Tycho Foerster, Maciej Pacut, and Stefan Schmid. "On the Complexity of Non-Segregated Routing in Reconfigurable Data Center Architectures". In: *ACM SIGCOMM Computer Communication Review* 49.2 (May 2019), pp. 2–8.

[191] Mowei Wang, Yong Cui, Shihan Xiao, Xin Wang, Dan Yang, Kai Chen, and Jun Zhu. "Neural Network Meets DCN: Traffic-Driven Topology Adaptation with Deep Learning". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2.2 (2018), pp. 1–25.

[192] Saim Salman, Christopher Streiffer, Huan Chen, Theophilus Benson, and Asim Kadav. "DeepConf: Automating Data Center Network Topologies Management with Machine Learning". In: *Proceedings of the 2018 Workshop on Network Meets AI & ML*. Budapest, Hungary: ACM, Aug. 2018, pp. 8–14.

[193] Albert Greenberg et al. "VL2: A Scalable and Flexible Data Center Network". In: *Proc. ACM SIGCOMM '09*. Barcelona, Spain, Aug. 2009, pp. 51–62.

[194] Min Yee Teh, Shizhen Zhao, and Keren Bergman. *METTEOR: Robust Multi-Traffic Topology Engineering for Commercial Data Center Networks*. 2020. arXiv: 2002.00473 [cs].

[195] Min Yee Teh, Shizhen Zhao, Peirui Cao, and Keren Bergman. "Enabling Quasi-Static Reconfigurable Networks With Robust Topology Engineering". In: *IEEE/ACM Transactions on Networking* (2022), pp. 1–15.

[196] Min Yee Teh, Zhenguo Wu, Madeleine Glick, Sebastien Rumley, Manya Ghobadi, and Keren Bergman. "Performance trade-offs in reconfigurable networks for HPC". In: *Journal of Optical Communications and Networking* 14.6 (May 2022), pp. 454–468.

[197] Hao Yang and Zuqing Zhu. "Topology configuration scheme for accelerating coflows in a hyper-FleX-LION". In: *Journal of Optical Communications and Networking* 14.10 (Sept. 2022), pp. 805–814.

[198] Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu. "SplitCast: Optimizing Multicast Flows in Reconfigurable Datacenter Networks". In: *Proc. IEEE INFOCOM '20*. Toronto, ON, Canada: IEEE, July 2020, pp. 2559–2568.

[199] Weiyang Wang et al. "TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs". In: *Proc. USENIX NSDI '23*. Boston, MA, USA: USENIX, Apr. 2023, pp. 739–767.

[200] Mehrdad Khani et al. "SiP-ML: high-bandwidth optical network interconnects for machine learning training". In: *Proc. ACM SIGCOMM '21*. Virtual: ACM, Aug. 2021, pp. 657–675.

[201] Alessandro Ottino, Joshua Benjamin, and Georgios Zervas. *RAMP: A Flat Nanosecond Optical Network and MPI Operations for Distributed Deep Learning Systems*. 2022. arXiv: 2211.15226 [cs.NI].

[202] Min Yee Teh, Yu-Han Hung, George Michelogiannakis, Shijia Yan, Madeleine Glick, John Shalf, and Keren Bergman. "TAGO: Rethinking Routing Design in High Performance Reconfigurable Networks". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. Atlanta, GA, USA: IEEE, Nov. 2020, pp. 1–16.

[203] Mamun Abu-Tair, Philip Perry, Philip Morrow, Sally McClean, Bryan Scotney, Gerard Parr, and Md Biswas. "Optical Space Switches in Data Centers: Issues with Transport Protocols". In: *Photonics* 6.1 (Feb. 22, 2019), pp. 1–16.

[204] Matthew K. Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C. Snoeren. "Adapting TCP for Reconfigurable Datacenter Networks". In: *17th USENIX Symposium on Networked Systems Design and Implementation*. Santa Clara, CA, USA: USENIX Association, Feb. 2020, pp. 651–666.

[205] Shawn Shuoshuo Chen, Weiyang Wang, Christopher Canel, Srinivasan Seshan, Alex C. Snoeren, and Peter Steenkiste. "Time-division TCP for reconfigurable data center networks". In: *Proc. ACM SIGCOMM '22*. Amsterdam Netherlands: ACM, Aug. 2022, pp. 19–35.

[206] Vamsi Addanki, Oliver Michel, and Stefan Schmid. "PowerTCP: Pushing the Performance Limits of Datacenter Networks". In: *19th USENIX Symposium on Networked Systems Design and Implementation*. Renton, WA, USA: USENIX Association, Apr. 2022, pp. 51–70.

[207] Klaus-Tycho Foerster, Manya Ghobadi, and Stefan Schmid. "Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures". In: *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems - ANCS '18*. Ithaca, New York, July 2018, pp. 89–96.

[208] Shizhen Zhao, Peirui Cao, and Xinbing Wang. "Understanding the Performance Guarantee of Physical Topology Design for Optical Circuit Switched Data Centers". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5.3 (Dec. 2021), pp. 1–24.

[209] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. "Optimal oblivious reconfigurable networks". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. Rome, Italy: ACM, June 2022, pp. 1339–1352.

[210] Tegan Wilson, Daniel Amir, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg. “Extending Optimal Oblivious Reconfigurable Networks to all N”. In: *Symposium on Algorithmic Principles of Computer Systems (APOCS)*. Florence, Italy: Society for Industrial and Applied Mathematics, Jan. 2023, pp. 1–16.

[211] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. “Xpander: Towards Optimal-Performance Datacenters”. In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. Irvine, CA, USA: ACM, Dec. 2016, pp. 205–219.

[212] Hasanin Harkous, Michael Jarschel, Mu He, Rastin Pries, and Wolfgang Kellerer. “Towards Understanding the Performance of P4 Programmable Hardware”. In: *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS*. Cambridge, United Kingdom: IEEE, Sept. 2019, pp. 1–6.

[213] Dominik Scholz, Henning Stubbe, Sebastian Gallenmuller, and Georg Carle. “Key Properties of Programmable Data Plane Targets”. In: *2020 32nd International Teletraffic Congress (ITC 32)*. Osaka, Japan: IEEE, Sept. 2020.

[214] Georgios P. Katsikas, Tom Barbette, Marco Chiesa, Dejan Kostić, and Gerald Q. Maguire. “What You Need to Know About (Smart) Network Interface Cards”. In: *Passive and Active Measurement*. Springer International Publishing, 2021, pp. 319–336.

[215] Amaury Van Bemten, Nemanja Deric, Amir Varasteh, Andreas Blenk, Stefan Schmid, and Wolfgang Kellerer. “Empirical Predictability Study of SDN Switches”. In: *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. Cambridge, United Kingdom: IEEE, Sept. 2019, pp. 1–13.

[216] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. “OFLOPS: An Open Framework for OpenFlow Switch Evaluation”. In: *Passive and Active Measurement*. Springer Berlin Heidelberg, 2012, pp. 85–95.

[217] Andreas Blenk, Arsany Basta, Laurenz Henkel, Johannes Zerwas, Wolfgang Kellerer, and Stefan Schmid. “perfbench”. In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. Budapest, Hungary: ACM, Aug. 2018.

[218] Bob Lantz, Brandon Heller, and Nick McKeown. “A network in a laptop”. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Monterey, California: ACM, Oct. 2010, pp. 1–6.

[219] Mehrdad Khani et al. “A Network Architecture for Fast Training of Distributed Machine Learning with Silicon Photonics”. In: (2019), pp. 1–21.

[220] Telcordia. *GR-831*. 1996. URL: https://telecom-info.njdepot.ericsson.net/site-cgi/ido/docs.cgi?ID=SEARCH%5C&DOCUMENT=GR-831 (visited on 06/30/2021).

[221] *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. Tech. rep. Oct. 2010. DOI: 10.17487/rfc6020.

[222] *Polatis Series 6000*. URL: https://www.polatis.com (visited on 06/30/2021).

[223] *FS.com 10GBASE-LR SFP+ 1310nm (SMF)*. URL: https://img-en.fs.com/file/datasheet/10g-base-lr-1310nm.pdf (visited on 06/30/2021).

[224] *Intel X710-DA4*. URL: https://www.intel.de/content/www/de/de/products/docs/network-io/ethernet/network-adapters/ethernet-x710-brief.html (visited on 06/30/2021).

[225] *Data Plane Development Kit*. URL: https://www.dpdk.org/ (visited on 04/24/2021).

[226] *Netronome - Agilio CX SmartNICs*. URL: https://www.netronome.com/products/agilio-cx/ (visited on 06/30/2021).

[227] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. "NetFPGA SUME: Toward 100 Gbps as Research Commodity". In: *IEEE Micro* 34.5 (Sept. 2014), pp. 32–41.

[228] *Pica P3297 Datasheet*. URL: https://www.pica8.com/wp-content/uploads/pica8-datasheet-48x1gbe-p3297.pdf (visited on 06/30/2021).

[229] *Dell S4048-ON Datasheet*. URL: https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-EMC-Networking-S4048-ON-Spec-Sheet.pdf (visited on 06/30/2021).

[230] *EdgeCore AS5835-54X-EC Datasheet*. URL: https://www.edge-core.com/_upload/images/AS5835-54X-EC_DS_R01_20210205.pdf (visited on 02/28/2023).

[231] *SONiC - Software for Open Networking in the Cloud*. URL: https://sonic-net.github.io/SONiC/ (visited on 02/28/2023).

[232] *iPerf*. 2020. URL: https://iperf.fr/ (visited on 06/30/2021).

[233] Ralf Kundel, Fridolin Siegmund, Jeremias Blendin, Amr Rizk, and Boris Koldehofe. "P4STA: High Performance Packet Timestamping with Programmable Packet Processors". In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. Budapest, Hungary, Apr. 2020, pp. 1–9.

[234] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. "MoonGen". In: *Proceedings of the 2015 Internet Measurement Conference*. Tokyo, Japan: ACM, Oct. 2015.

[235] A. Bierman, M. Bjorklund, and K. Watsen. *RESTCONF Protocol*. Tech. rep. Jan. 2017. DOI: 10.17487/rfc8040.

[236] Michael Grottke, Rivalino Matias, and Kishor S. Trivedi. "The fundamentals of software aging". In: *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)*. Seattle, WA, USA: IEEE, Nov. 2008.

[237] Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator". In: *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference*. Anaheim, CA, USA: USENIX, Apr. 2005, pp. 41–46. URL: http://www.usenix.org/events/usenix05/tech/freenix/bellard.html.

[238] Nvidia Corporation. *Nvidia Tesla T4 GPU Datasheet*. 2019. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf (visited on 11/26/2021).

[239] VNI Cisco. "Cisco visual networking index: Forecast and trends, 2017–2022". In: *White Paper* 1 (2018).

[240] Sushant Jain et al. “B4: Experience with a globally-deployed software defined WAN”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[241] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. “Achieving high utilization with software-driven WAN”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 15–26.

[242] Parveen Patel et al. “Ananta: Cloud scale load balancing”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 207–218.

[243] Daniel E Eisenbud et al. “Maglev: A fast and reliable software network load balancer”. In: *Proc. USENIX NSDI ’16*. Santa Clara, CA, USA, Mar. 2016, pp. 523–535.

[244] N Sambo, A Giorgetti, A Sgambelluri, E Riccardi, F Fresi, F Cugini, and P Castoldi. “Connection Setup in Open Systems: Interfaces, Amplification, and Equalization”. In: *2018 European Conference on Optical Communication (ECOC)*. Rome, Italy, Sept. 2018, pp. 1–3.

[245] Luis Velasco, Alberto Castro, Daniel King, Ori Gerstel, Ramon Casellas, and Victor Lopez. “In-Operation Network Planning”. en. In: *IEEE Commun. Mag.* 52.1 (Jan. 2014), pp. 52–60.

[246] *Coronavirus Disease (COVID-19) Situation Reports*. URL: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/situation-reports (visited on 05/18/2021).

[247] C. Labovitz. *Internet Traffic 2009-2019*. APRICOT 2019. 2019.

[248] Florian Wohlfart, Nikolaos Chatzis, Caglar Dabanoglu, Georg Carle, and Walter Willinger. “Leveraging Interconnections for Performance: The Serving Infrastructure of a Large CDN”. In: *Proc. ACM SIGCOMM ’18*. Hungary, Budapest, Aug. 2018, pp. 206–220.

[249] Vasileios Giotsas, Matthew Luckie, Bradley Huffaker, and kc claffy. “Inferring Complex AS Relationships”. In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, Nov. 2014, pp. 23–30.

[250] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. “Improving Content Delivery Using Provider-Aided Distance Information”. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. Melbourne, Australia: ACM, Nov. 2010, pp. 22–34.

[251] Benjamin Frank, Ingmar Poese, Georgios Smaragdakis, Steve Uhlig, and Anja Feldmann. *Content-Aware Traffic Engineering*. en. Feb. 2012. arXiv: 1202.1464 [cs].

[252] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. “On Dominant Characteristics of Residential Broadband Internet Traffic”. en. In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. Chicago, IL, USA: ACM, Nov. 2009, pp. 90–102.

[253] Jeffrey Erman and K.K. Ramakrishnan. “Understanding the Super-Sized Traffic of the Super Bowl”. en. In: *Proceedings of the 2013 conference on Internet measurement conference*. Barcelona, Spain: ACM, Oct. 2013, pp. 353–360.

[254] DE-CIX. *Big upswing in Internet usage due to Covid-19 measures*. https://www.de-cix.net/en/news-events/news/big-upswing-in-internet-usage-due-to-covid-19-measures. Accessed: 2020-06-22.

[255] Martin Birk et al. "Evolving to an SDN-Enabled Isp Backbone: Key Technologies and Applications". en. In: *IEEE Commun. Mag.* 54.10 (Oct. 2016), pp. 129–135.

[256] Akhilesh S. Thyagaturu, Anu Mercian, Michael P. McGarry, Martin Reisslein, and Wolfgang Kellerer. "Software Defined Optical Networks (SDONs): A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2738–2786.

[257] *Path Computation Element (PCE) Communication Protocol (PCEP)*. Tech. rep. Mar. 2009. DOI: 10.17487/rfc5440.

[258] *The State of Traffic Engineering - an ISP's Perspective*. APNIC Blog. Aug. 23, 2018. URL: https://blog.apnic.net/2018/08/24/the-state-of-traffic-engineering-an-isps-perspective/ (visited on 05/19/2021).

[259] L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. *Segment Routing Architecture*. Tech. rep. July 2018. DOI: 10.17487/rfc8402.

[260] E. Crabbe, I. Minei, J. Medved, and R. Varga. *Path Computation Element Communication Protocol (PCEP) Extensions for Stateful PCE*. Tech. rep. Sept. 2017. DOI: 10.17487/rfc8231.

[261] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. "End-user mapping: Next generation request routing for content delivery". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 167–181.

[262] Kok-Kiong Yap et al. "Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering". In: *Proc. ACM SIGCOMM '17*. Los Angeles, CA, USA, Aug. 2017, pp. 432–445.

[263] Brandon Schlinker et al. "Engineering Egress with Edge Fabric: Steering Oceans of Content to the World". In: *Proc. ACM SICOMM '17*. Los Angeles, CA, USA, Aug. 2017, pp. 418–431.

[264] Bruce M. Maggs and Ramesh K. Sitaraman. "Algorithmic Nuggets in Content Delivery". In: *ACM SIGCOMM Computer Communication Review* 45.3 (2015), pp. 52–66.

[265] S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Woundy. *Application-Layer Traffic Optimization (ALTO) Protocol*. Tech. rep. Sept. 2014. DOI: 10.17487/rfc7285.

[266] Giovane CM Moura, John Heidemann, Ricardo de O Schmidt, and Wes Hardaker. "Cache me if you can: effects of DNS time-to-live". In: *Proceedings of the Internet Measurement Conference*. Amsterdam, Netherlands: ACM, Oct. 2019, pp. 101–115. DOI: 10.1145/3355369.3355568.

[267] Krzysztof Walkowiak and Miroslaw Klinkowski. "Joint Anycast and Unicast Routing for Elastic Optical Networks: Modeling and Optimization". en. In: *2013 IEEE International Conference on Communications (ICC)*. Kuala Lumpur, Malaysia: IEEE, June 2013, pp. 3909–3914.

[268] Srivatsan Balasubramanian, Satyajeet Ahuja, Gaya Nagarajan, Andrea Celletti, and Frantisek Foston. "Multilayer Planning for Facebook Scale Worldwide Network". en. In: *2017 International Conference on Optical Network Design and Modeling (ONDM)*. Budapest, Hungary: IEEE, May 2017, pp. 1–6.

[269] Xin Jin et al. "Optimizing bulk transfers with software-defined optical WAN". In: *Proc. ACM SIGCOMM '16*. ACM. Florianopolis, Brazil, Aug. 2016, pp. 87–100.

[270] Matthias Wichtlhuber, Robert Reinecke, and David Hausheer. “An SDN-Based CDN/ISP Collaboration Architecture for Managing High-Volume Flows”. en. In: *IEEE Trans. Netw. Serv. Manage.* 12.1 (Mar. 2015), pp. 48–60. ISSN: 1932-4537.

[271] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. “Efficiently Delivering Online Services over Integrated Infrastructure”. en. In: *Proc. USENIX NSDI '16*. Santa Clara, CA, USA, Mar. 2016, pp. 77–90.

[272] Mostafa Ammar, Ellen Zegura, and Yimeng Zhao. “A Vision for Zero-Hop Networking (ZeN)”. en. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Atlanta, GA, USA, June 2017, pp. 1765–1770.

[273] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. “Cooperative Content Distribution and Traffic Engineering in an ISP Network”. en. In: *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. Seattle, WA, USA: ACM, June 2009, pp. 239–250.

[274] Abhigyan Sharma, Arun Venkataramani, and Ramesh K Sitaraman. “Distributing Content Simplifies ISP Traffic Engineering”. en. In: *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*. Pittsburgh, PA, USA: ACM, June 2013, p. 14.

[275] P. Papanikolaou, K. Christodoulopoulos, and E. Varvarigos. “Incremental Planning of Multi-Layer Elastic Optical Networks”. en. In: *Proc. International Conference on Optical Network Design and Modeling (ONDM) 2017*. Budapest, Hungary, May 2017, pp. 1–6.

[276] Jennifer Gossels, Gagan Choudhury, and Jennifer Rexford. “Robust Network Design for IP/Optical Backbones”. en. In: *J. Opt. Commun. Netw.* 11.8 (Aug. 2019), pp. 478–490.

[277] Nicolas Herbaut, Daniel Negru, Yiping Chen, Pantelis A. Frangoudis, and Adlen Ksentini. “Content Delivery Networks as a Virtual Network Function: A Win-Win ISP-CDN Collaboration”. en. In: *2016 IEEE Global Communications Conference (GLOBECOM)*. Washington, DC, USA: IEEE, Dec. 2016, pp. 1–6.

[278] Khaled Diab and Mohamed Hefeeda. “Joint Content Distribution and Traffic Engineering of Adaptive Videos in Telco-CDNs”. en. In: *Proc. IEEE INFOCOM '19*. Paris, France: IEEE, Apr. 2019, pp. 1342–1350. ISBN: 978-1-72810-515-4. DOI: 10.1109/INFOCOM.2019.8737635.

[279] Zilong Ye, Xiaojun Cao, Jianping Wang, Hongfang Yu, and Chunming Qiao. “Joint Topology Design and Mapping of Service Function Chains for Efficient, Scalable, and Reliable Network Functions Virtualization”. en. In: *IEEE Network* 30.3 (May 2016), pp. 81–87.

[280] *IBM ILOG CPLEX Optimization Studio 12.9*. https://www.ibm.com/products/ilog-cplex-optimization-studio.

[281] Thomas Favale, Francesca Soro, Martino Trevisan, Idilio Drago, and Marco Mellia. “Campus traffic and e-Learning during COVID-19 pandemic”. In: *Computer Networks* 176 (July 2020), p. 107290. DOI: 10.1016/j.comnet.2020.107290.

[282] BBC. *Netflix to cut streaming quality in Europe for 30 days*. https://www.bbc.com/news/technology-51968302. Accessed: 2020-06-22.

[283] Sandvine. *Over 43% of the Internet Is Consumed by Netflix, Google, Amazon, Facebook, Microsoft, and Apple: Global Internet Phenomena Spotlight*. URL: https://www.sandvine.com/blog/netflix-vs.-google-vs.-amazon-vs.-facebook-vs.-microsoft-vs.-apple-traffic-share-of-internet-brands-global-internet-phenomena-spotlight (visited on 05/14/2021).

[284] Andreas Blenk, Patrick Kalmbach, Wolfgang Kellerer, and Stefan Schmid. "o'zapft is: : Tap Your Network Algorithm's Big Data!" In: *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. Los Angeles, CA, USA: ACM, Aug. 2017, pp. 19–24.

[285] Tom Leighton. "Improving performance on the internet". In: *Communications of the ACM* 52.2 (2009), pp. 44–51.

[286] Tobias Flach et al. "Reducing web latency: the virtue of gentle aggression". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 159–170.

[287] Matthias Wichtlhuber, Jan Kessler, Sebastian Bucker, Ingmar Poese, Jeremias Blendin, Christian Koch, and David Hausheer. "SoDA: Enabling CDN-ISP Collaboration with Software Defined Anycast". en. In: *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. Stockholm, Sweden: IEEE, June 2017, pp. 1–9.

[288] *Google Fiber Inc.* URL: https://fiber.google.com/ (visited on 06/30/2020).

[289] *Amazon receives FCC approval for Project Kuiper satellite constellation*. URL: https://blog.aboutamazon.com/company-news/amazon-receives-fcc-approval-for-project-kuiper-satellite-constellation (visited on 06/30/2020).

[290] *ATT TV*. URL: https://www.att.com/tv/ (visited on 06/30/2020).

[291] *Xfinity Digital Cable TV*. URL: https://www.xfinity.com/learn/digital-cable-tv (visited on 06/30/2020).

[292] Brigitte Jaumard, Huy Quang Duong, Romualdas Armolavicius, Todd Morris, and Petar Djukic. "Efficient real-time scalable make-before-break network re-routing". In: *IEEE/OSA Journal of Optical Communications and Networking* 11.3 (2019), pp. 52–66.

[293] Pierre François, Clarence Filsfils, John Evans, and Olivier Bonaventure. "Achieving sub-second IGP convergence in large IP networks". In: *Comput. Commun. Rev.* 35.3 (2005), pp. 35–44.

[294] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. "A throughput-centric view of the performance of datacenter topologies". In: *Proc. ACM SIGCOMM '21*. Virtual Event: ACM, Aug. 2021, pp. 349–369.

[295] M. Frans Kaashoek and David R. Karger. *Koorde: A Simple Degree-Optimal Distributed Hash Table*. 2003. DOI: 10.1007/978-3-540-45172-3_9.

[296] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. "Chord: a scalable peer-to-peer lookup protocol for internet applications". In: *IEEE/ACM Transactions on Networking* 11.1 (Feb. 2003), pp. 17–32.

[297] Mohammad Hosseinabady, Mohammad Reza Kakoee, Jimson Mathew, and Dhiraj K. Pradhan. "Low Latency and Energy Efficient Scalable Architecture for Massive NoCs Using Generalized de Bruijn Graph". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19.8 (Aug. 2011), pp. 1469–1480.

[298] Yiou Chen, Jianhao Hu, and Xiang Ling. “De Bruijn graph based 3D Network on Chip architecture design”. In: *2009 International Conference on Communications, Circuits and Systems*. Milpitas, CA, USA: IEEE, July 2009, pp. 986–990.

[299] Peyman Faizian, Md Atiqul Mollah, Xin Yuan, Zaid Alzaid, Scott Pakin, and Michael Lang. “Random Regular Graph and Generalized De Bruijn Graph with k-Shortest Path Routing”. In: *IEEE Transactions on Parallel and Distributed Systems* 29.1 (Jan. 2018), pp. 144–155.

[300] K.N. Sivarajan and R. Ramaswami. “Lightwave networks based on de Bruijn graphs”. In: *IEEE/ACM Transactions on Networking* 2.1 (1994), pp. 70–79.

[301] Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. “A cost comparison of datacenter network architectures”. In: *Proceedings of the 6th International COnference*. Philadelphia, PA, USA: ACM, Nov. 2010.

[302] Frank Dürr. *A Flat and Scalable Data Center Network Topology Based on De Bruijn Graphs*. 2016. arXiv: 1610.03245 [cs.NI].

[303] Nicolaas Govert De Bruijn. “A combinatorial problem”. In: *Proc. Koninklijke Nederlandse Academie van Wetenschappen* 49 (1946), pp. 758–764.

[304] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures Arrays · Trees · Hypercubes. Arrays · Trees · Hypercubes*. Elsevier Science & Technology Books, 2014. ISBN: 9781483221151.

[305] Jon Kleinberg. “The small-world phenomenon. an algorithmic perspective”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. Portland, OR, USA: ACM, May 2000, pp. 163–170.

[306] Petar Maymounkov and David Mazières. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”. In: *Peer-to-Peer Systems, First International Workshop, IPTPS 2002*. Vol. 2429. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Mar. 2002, pp. 53–65. DOI: 10.1007/3-540-45748-8.

[307] P. Hall. “On Representatives of Subsets”. In: *Journal of the London Mathematical Society* s1-10.1 (Jan. 1935), pp. 26–30. DOI: 10.1112/jlms/s1-10.37.26.

[308] Harold N. Gabow. “Data Structures for Weighted Matching and Extensions to b-matching and f-factors”. In: *ACM Trans. Algorithms* 14.3 (2018), 39:1–39:80.

[309] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. “SketchLib: Enabling Efficient Sketch-based Monitoring on Programmable Switches”. In: *Proc. 19th USENIX NSDI '22*. Renton, WA, USA, Apr. 2022, pp. 743–759.

[310] Bala Kalyanasundaram and Kirk Pruhs. “Online Weighted Matching”. In: *Journal of Algorithms* 14.3 (May 1993), pp. 478–488.

[311] Mohammad Alizadeh et al. “Data center TCP (DCTCP)”. In: *Proc. ACM SICOMM '10*. New Delhi, India: ACM, Aug. 2010, pp. 63–74.

[312] Vojislav Dukic, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. “Is advance knowledge of flow sizes a plausible assumption?” In: *Proc. 16th USENIX NSDI '19*. Boston, MA, USA: USENIX Association, Feb. 2019, pp. 565–580.

[313] Chuanxiong Guo et al. “BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers”. In: *Proc. ACM SIGCOMM '09*. Barcelona, Spain, Aug. 2009, pp. 63–74.

[314] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. “Jellyfish: Networking Data Centers Randomly”. In: *Proc. USENIX NSDI '12*. San Jose, CA, USA, Apr. 2012, pp. 1–14.

[315] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. “Flyways To De-Congest Data Center Networks”. In: *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09*. New York City, NY, USA: ACM SIGCOMM, Oct. 2009, pp. 1–6.

[316] Sangeetha Abdu Jyothi, Ankit Singla, Brighten Godfrey, and Alexandra Kolla. “Measuring and understanding throughput of network topologies”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Salt Lake City, UT, USA: IEEE Computer Society, Nov. 2016, pp. 761–772.

[317] Haitao Wu, Guohan Lu, Dan Li, Chuanxiong Guo, and Yongguang Zhang. “MDCube: a high performance network structure for modular data center interconnection”. In: *Proceedings of the 2009 ACM Conference on Emerging Networking Experiments and Technology*. Rome, Italy: ACM, Dec. 2009, pp. 25–36.

[318] Chen Avin and Stefan Schmid. “ReNets: Statically-Optimal Demand-Aware Networks”. In: *Symposium on Algorithmic Principles of Computer Systems (APOCS)*. Virtual: Society for Industrial and Applied Mathematics, Jan. 2021, pp. 25–39.

[319] Oded Goldreich. “Basic Facts about Expander Graphs”. In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. Ed. by Oded Goldreich. Vol. 6650. Lecture Notes in Computer Science. Springer, 2011, pp. 451–464. DOI: 10.1007/978-3-642-22670-0_{3}{0}.