Lehrstuhl für Netzwerkarchitekturen
Fakultät für Informatik
Technische Universität München

Traffic-Adaptive Routing

Nils Kammenhuber

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München
zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Chr. Scheideler
Prüfer der Dissertation:	1.	Univ.-Prof. A. Feldmann, Ph. D., Technische Universität Berlin
	2.	Univ.-Prof. Dr. Th. Fuhrmann
	3.	Prof. B. M. Maggs, Ph. D., Carnegie Mellon University (schriftliche Beurteilung)

Die Dissertation wurde am 19.12.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.06.2008 angenommen.

Abstract

Traffic in the Internet is known to change over time and exhibit volatile behaviour. One major challenge in communication networks is the problem of handling these traffic demands that are bursty and hard to predict. Current traffic engineering techniques operate on time scale of several hours, which is too slow to react to quick phenomena such as flash crowds or BGP reroutes. The obvious solution, load sensitive routing, is frowned upon, since routing decisions at short time scales can lead to oscillations. This has prevented load sensitive routing from being deployed since the early experiences in Arpanet, the predecessor of today's Internet.

However, recent theoretical results have shown that a re-routing policy based on game theory provably avoids such oscillation and, in addition, can be shown to converge quickly. In the main part of our work, we describe REPLEX, a distributed dynamic traffic engineering algorithm based on this policy. Exploiting the fact that most underlying routing protocols support multiple equal-cost routes to a destination, it dynamically changes the proportion of traffic that is routed along each path. These traffic proportions are carefully adapted utilising information from periodic measurements and information exchanged between the routers. The required signalling overhead is small. Moreover, it can, under some circumstances, be avoided altogether.

We evaluate our algorithm in extensive simulations employing traffic loads that mimic realistic bursty TCP traffic whose characteristics are consistent with self-similarity. The simulations show quick convergence and no oscillations with reasonably chosen parameters. Comparative simulations on realistic network topologies show that REPLEX achieves performance improvements that are as good, if not better than those attained with traditional traffic engineering methods.

Since REPLEX as well as many other applications rely on collecting traffic statistics, we furthermore describe an algorithm that enables a router to perform this task in an efficient way. Our Expand-and-Collapse algorithm (EaC) is a heuristic that is particularly aimed at platforms like network processors: Under severe memory constraints, it finds the most popular paths in search trees such as those are used in most packet classification algorithms. EaC allows to efficiently find the largest traffic contributors. In addition, can be applied in a wide number of other contexts as well, due to its genericity. Our theoretical and simulation-based analyses show good performance.

A sound understanding for the behaviour of the workload traffic is a vital factor in the construction of load-adaptive dynamic routing and traffic engineering systems. We therefore also describe two methodologies for studying the characteristics of Web traffic, which is still a major part of today's Internet traffic. Our contributions involve on the one hand a methodology for estimating Web traffic demands on a global scale,

through the combination of local measurements and log files from a content distribution network. The traffic matrices we derived are among the largest of their kind and reveal changes over time that grow larger as the time elapses. The other method that we present is a model-based analysis of Web search traffic and the Web traffic imposed by subsequent clicks that follow the search result. Our analysis confirms previous findings and reveals new interesting aspects of Web search related user behaviour.

Zusammenfassung

Es ist allgemein bekannt, dass Datenverkehr im Internet zeitlichen Änderungen und stark schwankendem Verhalten unterworfen ist. Eine der vordringlichsten Herausforderungen beim Betrieb von Datennetzwerken stellt daher der Umgang mit stoßartig wechselndem und kaum voraussagbaren Verkehrsaufkommen dar. Heutige Methoden des Traffic Engineering (deutsch: Verkehrslenkung) reagieren auf Verkehrsänderungen binnen einiger Stunden, was zum Gegensteuern bei schnell auftretenden Phänomenen wie Flash Crowds oder Instabilitäten von BGP nicht ausreichend ist. Die naheliegende Lösung, das Routing lastabhängig zu gestalten, ist jedoch allgemein verpönt: Interagierende Routingentscheidungen auf kurzen Zeitskalen können zu Oszillationen führen – ein Effekt, der bereits im Arpanet, dem Vorgänger des heutigen Internets, beobachtet wurde, und aufgrunddessen der Einsatz lastabhängigen Routings heute gemieden wird.

Kürzlich erschienene theoretische Arbeiten haben jedoch eine spieltheoretische Re-Routing-Strategie aufgezeigt, die solche Oszillationen beweisbar vermeidet, und von der sich darüberhinaus zeigen lässt, dass sie schnell konvergiert. Im Hauptteil unserer Arbeit beschreiben wir REPLEX, einen auf dieser Strategie basierenden verteilten dynamischen Traffic-Engineering-Algorithmus. Durch Ausnutzen der Tatsache, dass viele Routingprotokolle zu einem Ziel mehrere gleichberechtigte Routen mit äquivalenten Kosten zulassen (englisch: multipath equal-cost routes), passt REPLEX die jeweiligen Anteile der einzelnen Routen am Gesamtaufkommen eines Datenstroms zu einem Ziel dynamisch an. Die entsprechenden Verhältnisse werden permanent auf Basis periodischer Messungen sowie zwischen den Routern ausgetauschten Informationen angepasst. Der hierfür erforderliche zusätzliche Informationsaufwand ist klein und kann darüberhinaus unter bestimmten Bedingungen sogar vollständig entfallen.

Wir evaluieren unseren Algorithmus umfassend mit Hilfe von Netzwerksimulationen. Dabei verwenden wir Prüflasten, die realistischen und stark schwankenden TCP-Verkehr simulieren, und der konsistent mit selbstähnlichem Verhalten ist. Unsere Simulationen zeigen schnelle Konvergenz und das Ausbleiben von Oszillationen bei Verwendung vernünftiger Parameter für unseren Algorithmus. Vergleichsmessungen auf realistischen Netzwerktopologien zeigen, dass die durch REPLEX ermöglichten Leistungsverbesserungen mindestens so gut oder sogar besser sind als die mit traditionellen Traffic-Engineering-Verfahren erreichbaren.

Da sowohl REPLEX als auch viele andere Anwendungen auf das Sammeln von Verkehrsstatistiken angewiesen sind, präsentieren wir desweiteren einen Algorithmus, mit dem diese Aufgabe auf effiziente Art und Weise von einem Router durchgeführt werden kann. Unser Expand-and-Collapse-Algorithmus (EaC) ist eine insbesondere auf

Architekturen wie Netzwerkprozessoren ausgerichtete Heuristik, welche unter sehr restriktiven Speichervoraussetzungen die populärsten Pfade in Suchbäumen, welche typischerweise in Paketklassifikationsalgorithmen zur Anwendung kommen, findet. Er erlaubt hierdurch ein effizientes Auffinden der dominierenden Verkehrsbeiträger, kann jedoch aufgrund seiner Allgemeinheit auch in völlig anderen Kontexten zur Anwendung kommen. Sowohl theoretische als auch simulationsbasierte Analysen zeigen eine gute Performance.

Eine Grundvoraussetzung für den Entwurf dynamischer lastabhängiger Routing- und Traffic-Engineering-Systeme ist ein tiefgreifendes Verständnis des zu routenden Verkehrs. Daher beschreiben wir desweiteren zwei Analysemethoden für WWW-Verkehr, welcher auch heute noch einen großen Anteil am Gesamtverkehrsaufkommen stellt. Unsere Beiträge umfassen zum Einen eine Methodik zur Abschätzung von Webverkehrsströmen auf weltweiter Ebene, wobei eine Kombination lokaler Messungen und Logdateien eines Content-Distribution-Netzwerks zum Einsatz kommt. Die von uns erstellten Verkehrsmatrizen können mit zu den größten ihrer Art gezählt werden und weisen eine zeitliche Variabilität auf, welche mit wachsendem zeitlichem Abstand zunehmend größer wird. Die andere von uns präsentierte Methode ist eine modellbasierte Analyse von Web-Suchanfragen und den einer Suchanfrage nachfolgenden Klicks. Unsere Analyse bestätigt frühere Ergebnisse und liefert interessante neue Aspekte bezüglich des Nutzerverhaltens im Kontext von Websuchmaschinen.

Notation and Usage

Typographic Conventions

- Words from program code, language keywords, file names, program calls, commands, console input, console output etc. are printed in `typewriter` font.
- Whenever a notion is introduced and explained, it is printed in *italics*, and it will also appear on the margin. This occurrence will show up in the index in boldface.
- Software and other registered or trademarked products are written in a sans-sérif font when they are introduced.

Index Usage

- An index entry "A, *see* B→C" means that for looking up *A*, the reader should look up the entry *B* and consult its sub-entry *C*.
- Index entries that point to footnotes are marked with *n* preceding the page number.
- Index entries that point to figures are marked with *f* preceding the page number.
- Index entries that point to the glossary are marked with *g* preceding the page number.
- Index entries that are printed in boldface point to the definition or explanation of the expression in question.

Spelling

We shall use British spelling throughout this work. For example, we write *centred* and *fibre* (not *centered* or *fiber*); we write *naïve* instead of naive; we write *optimise*, *colour* and *acknowledgement* instead of *optimize*, *color* and *acknowledgment*, etc. Exceptions to this are made for the spelling of established terms, trademarks, or notions, e. g., *routing* instead of *routeing*.[1]

[1] In British English, *routing* without an *e* actually refers to drilling large holes into pieces of wood — an aspect which this thesis admittedly does not cover to a great extent.

License

Contents

1 Introduction

In the past decade, the Internet has had a tremendous impact on many people's lives, as it has greatly facilitated the process of communicating with other humans (e. g., via e-mail or Internet telephony), the process of searching and obtaining information (e. g., Web pages and search engines), and facilitated new appliances and amenities (e. g., file-sharing, cooperation-based projects like Wikipedia or open source software development, online games, telemedicine, online auctions, online forums). In other words, the Internet has become and is still becoming more and more important for our daily life. However, judging by its history, it is apparent that new applications and services tend to require ever more and more network resources—just compare the data volumes of e-Mail (1980s) to those of the Web (1990s), to file sharing (since 2000), or to video-on-demand (today). Obviously, the Internet's infrastructure does not only need to constantly evolve in *quantity* (in that more and more people gain Internet access), but also in *quality* (in that the network can satisfy the ever-growing demands of new applications).

This need for constant improvement comes at a price, of course, and that is the price of new hardware. However, faster routers and faster line cards do not automatically guarantee an improvement in network performance. Rather, their use has to be carefully planned, since providers, like any company, want to use their resources as efficiently as possible. An important methodology to ensure that the hardware is used in an efficient way is *traffic engineering*—meaning to optimise the routing, i. e., the way that data packets travel through the provider's network. This process can be imagined as planning how to set direction signposts in a road network (where all cars strictly follow these signposts), with the goal to maximise the road network's capacity (i. e., the number of cars that can travel through the network during some time interval), or to minimise traffic jams, or to minimise the travel times through the network.

Obviously, optimal routing depends on how much traffic is flowing from where to where in the network, i. e., the *traffic demands*: When travelling from Saarbrücken to Munich, it can be advisable to travel either via Karlsruhe or via Mannheim; the optimality of the choice depends on the traffic conditions on the different paths—which may change during different times of day. Similar to road networks, traffic demands in data networks are constantly subject to changes. These changes may, on the one hand, occur on long-range timescales. This is the case, e. g., for circadian effects or day-of-week changes. Such slow and usually repeating patterns can be anticipated, and the network operator can provide different routing setups for, e. g., different times of the day, and different days of the week.

traffic demand

On the other hand, not all changes in traffic demand are predictable; worse yet, these unpredicted changes may bring about drastic changes in the imposed demands within

relatively short periods of time. Flash crowds or similar phenomena affect user behaviour and their imposed demands on a short timescale. Similar effects can be the result of the release of new worms or viruses, denial-of-service attacks, or other malicious causes [DFM+06, Pax99]. Other changes include BGP re-routes and other technical causes [RWXZ02, LAJ99, LMJ99, LMJ98, GR97, SKDV00, SDV02, GP01, GW99, MGWR01]. Therefore, mechanisms that can dynamically react to such changes are favourable over static methods.

Nevertheless, most methods employed by network operators today are of a static or quasi-static nature: The operator collects traffic measurements from the network, centrally computes the traffic demands, calculates a new routing setup based on these demands, and installs the new routing setup in the network. Even though this process can be automatised, it still a static method: The system cannot react to sudden dramatic shifts in traffic happening on a timescale of minutes or seconds, which would require immediate action—due to a number of technical reasons which we will explain later, this process should rather be executed in large-scale time intervals.

Recently, however, different algorithms and mechanisms have been proposed that enable traffic engineering in a dynamic fashion. Following this development, in the main part of our thesis (chapters 6 and 7), we present REPLEX, a system that allows to do traffic engineering that adjusts to changes in traffic demands very quickly. Our algorithm is based on strong game-theoretic results, and it provides a number of interesting features, such as independence from and therefore full compatibility to all kinds of existing routing infrastructures that are used in today's networks. In a REPLEX-enabled network, all routers (or a subset of the routers) run an instance of REPLEX. The core idea is that every REPLEX router can dynamically shift slices of traffic between routes of equal cost. These equal-cost routes can be provided by existing routing protocols (e. g., OSPF, IS-IS, etc.), or they can be configured manually. The REPLEX-enabled routers exchange information on network traffic conditions through an efficient protocol that minimises overhead. We present results from extensive simulation-based analyses of REPLEX that draw on realistic self-similar TCP traffic. These show that our algorithm yields good improvements in network performance while converging quickly without causing oscillations, and demonstrate conformance with previous theoretical results. We furthermore show through simulations that its performance is comparable to, or even exceeds that of traditional traffic engineering in realistic topologies.

Obviously, dynamic traffic engineering methods like REPLEX, as well as other applications, require knowledge about traffic demands in the network. Obtaining the demands from network interfaces is an active area in network research (traffic matrix estimation). In this work, we present an approach for locally finding large traffic contributors in a memory-efficient way: our Expand-and-Collapse algorithm (EaC; chapter 5). It is especially suitable for platforms like network processors, but due to its genericity can be applied on a wide range of platforms—and even in completely different contexts such as data mining.

The basic idea of EaC is as follows. Every time a router receives a data packet, it needs to classify this packet, in order to decide what it needs to do with the packet. In many cases, this classification is performed using a search tree based algorithm of some sort. Although it is possible to simply keep a counter for each packet class, such an approach would require a large amount of fast memory for the individual counters. EaC instead augments the search tree by dynamically placing a small number of counters within the search tree, and thus drastically reduces the consumption of fast memory for the counters.

In order to be able to judge the effectiveness of load-adaptive routing and dynamic traffic engineering techniques, it is vital to explore the characteristics of the traffic that crosses the network. Estimating intradomain traffic (i. e., traffic demands within a service provider's own network) has thus been the topic of intense research over the past couple of years.

However, to the best of our knowledge, there has been no good methodology for estimating interdomain traffic matrices, i. e., global traffic demands between different independent networks. It is our understanding that even the question of whether interdomain traffic matrices and intradomain traffic demand matrices have similar dynamics remains unanswered. As Web traffic today is still one of the major contributors of Internet traffic, we present a technique that allows to measure interdomain Web traffic demands (i. e., traffic demands between autonomous networks, e. g., of independent Internet service providers) on a global scale, and that can furthermore be used to investigate spatial as well as time effects in the demands (chapter 3).

A specific kind of Web traffic is the traffic that is related to search engines. As search engines are one of the key features of today's WWW and form a vital part of the infrastructure of today's information age, a better understanding of their influence on Web traffic can be useful in a variety of ways. Although search engines form an active research area in itself, most of the research focuses on pure search engine log files, and only very little research investigates the relation of search engine results with accesses to pages that are directly or indirectly (i. e., via one ore more clicks in between) reachable from the search result list. We present a state model that captures these relations (chapter 4). Using this model on search logs extracted from packet level traces, we present a number of interesting insights pertaining to user browsing behaviour following the search result list.

Overview

The remaining chapters of this document contain the following information:

Chapter 2 contains background information: It gives an overview on how routing is done in the Internet, presents some of the most important routing protocols, and describes some aspects of traditional traffic engineering.

Chapter 3 describes a technique for estimating interdomain Web traffic demands, and presents results gained from applying this technique to a large data set.

Chapter 4 describes a methodology for analysing Web searches using a state model, and presents results from applying this model to data obtained from live packet level traces.

Chapter 5 presents the Expand-and-Collapse Algorithm (EaC), which allows to find a search tree's most popular nodes in a resource efficient way, and which routers can use to gather statistics about the traffic distribution of oncoming traffic.

Chapter 6 introduces REPLEX, a game-theoretic algorithm for dynamic traffic engineering that is applicable in a wide range of scenarios. We begin with a theoretical model from game theory, which we then evolve into an algorithm that can be applied in real-world IP networks.

Chapter 7 presents an extensive simulation-based evaluation of REPLEX using realistic traffic sources. We start with simulations for finding good settings for REPLEX, and then compare the performance improvements achieved by REPLEX to those of traditional traffic engineering.

Chapter 8 concludes the thesis and provides an outlook and open questions.

Published Work

Parts of this thesis have been published:

Chapter 3:

Anja Feldmann, Nils Kammenhuber, Olaf Maennel, Bruce Maggs, Roberto De Prisco, Ravi Sundaram: *A Methodology for Estimating Interdomain Web Traffic Demand.* Proceedings of the ACM Internet Measurement Conference (IMC), Taormina, Italy, 2004

Anja Feldmann, Nils Kammenhuber, Olaf Maennel, Bruce Maggs, Ravi Sundaram: *A methodology for estimating interdomain Web traffic demand.* Technical Report TUM-I0305, Technische Universität München, Munich, Germany, 2003

Chapter 4:

Nils Kammenhuber, Julia Luxenburger, Anja Feldmann, Gerhard Weikum: *Web Search Clickstreams.* Proceedings of the ACM Internet Measurement Conference (IMC), Rio de Janeiro, Brazil, 2006

Chapter 5:

Lukas Kencl, Nils Kammenhuber: *Efficient Statistics Gathering from Tree-Search Methods in Packet Processing Systems.* Proceedings of the IEEE International Conference in Communications (ICC), Seoul, Korea, 2005

Nils Kammenhuber, Lukas Kencl: *Efficient Statistics Gathering from Tree-Search Methods in Packet Processing Systems.* Intel Research Report No. IRC-TR-04-034, Cambridge, United Kingdom, September 2004

Chapter 6:

Simon Fischer, Nils Kammenhuber, Anja Feldmann: REPLEX—*Dynamic Traffic Engineering Based on Wardrop Routing Policies.* Proceedings of the 2nd Conference on Future Networking Technologies (CoNEXT), Lisbon, Portugal, 2006

2 Background

Now that the general direction of this work has been explained, we first describe the background and other works that are related to the concepts and methods presented in this thesis. As was laid out, the ultimate focus of the thesis is on dynamic optimisation of routing in the Internet, also referred to as dynamic traffic engineering. We thus give a brief overview on routing and traffic engineering in the Internet.

2.1 Routing

Data to be sent across the Internet is partitioned into in data packets, which then are individually transmitted across the network. The transmission of each packet is independent of other packets; the setup of a connection (such as in e. g., the telephone network) is therefore not needed. With regards to individual end-to-end connections, the Internet is thus a *stateless* network.

stateless

The format of the packets adheres to a certain format: the Internet Protocol (*IP*) specification [Pos81]. IP is the "glue" of the Internet: Figuratively speaking, it forms a greatest common denominator for communication; all hosts that together form the Internet can understand it—regardless of the underlying communication infrastructure (Ethernet, ATM, mobile phone networks, wireless LAN, etc.). IP packets can be thought of as standardised freight containers for data: Just as a freight container can be transported on a lorry, on a ship or via aeroplane, there is no need to rearrange its content when a freight container (or an IP packet, respectively) is moved onto another transport medium during its journey towards its destination.

IP

2.1.1 IP addresses

Similar to postal packets or freight containers, each IP packet carries the address of its source and its destination in its header, together with other information such as, e. g., the size of the packet. The source and destination addresses are called *IP addresses*.[1]

IP address

As with the postal system or the telephone network, an Internet host sending a packet to another host is normally not required to know the entire path that the packet will travel through the network—rather, it forwards he packet to a neighbouring network node (referred to as a *router*) that is situated closer towards the destination. This router then *forwards* the packet to the next router it deems to be on a valid path towards the

router

forwarding

[1] In today's Internet, these are mostly 4-byte values (so-called IP version 4, *IPv4*); whereas its successor IP version 6 (*IPv6*) with 16-byte addresses [SD98] is gaining increasing popularity and is meant to replace IPv4 in the long run.

destination. Note that the IP packet header normally does not prescribe the exact path along which the packet has to travel—rather, the decision on which of its neighbouring routers the packet will be sent to is typically based on its destination address. Sometimes, a router also takes other attributes of the packet into consideration, e. g., the type of data being transmitted, or the source address.

2.1.2 Prefixes and routes

In this system, the routers obviously need to somehow obtain knowledge regarding which packets should be sent along which link. Of course, with the millions of end hosts that are connected to the Internet, it is completely infeasible to store this information for each individual IP address. Instead, IP addresses are grouped into network *prefixes*: To this end, they are regarded as bit vectors, and all addresses that have the same prefix of bits (starting from bit 0 and going to some bit ℓ) are treated the same way; the idea is that hosts and routers in the same network area are assigned IP addresses from the same prefix range. Prefixes can have different bit lengths[2]. A common notation for prefixes is to treat those IP address bits that are not relevant to the prefix as zeroes, convert the prefix to a human-readable form, and add the prefix length ℓ as `/`ℓ (e. g., `131.159.14.0/25`).

prefix

Recall that standard IP routing is based on IP destination addresses. When a router receives a packet to be forwarded towards some destination, it thus inspects the packet's destination address and compares it against a vector of *routes* that it knows. A route, in its simplest case, is a prefix acting as a mask that tells which packets should be forwarded via this route, and a reference to an interface (*NIC*; *network interface card*) connected to a neighbouring router. The vector of routes is referred to as a *forwarding table*, or the router's (general) *routing table*. The first prefix that matches the packet's destination address determines thus what the router will do with the packet.

route

NIC

forwarding table

routing table

IP prefixes (and thus, routes) can be organised hierarchically, thereby allowing different levels of details across the network, as well as exceptions. For example, a router in the U. S. is not required to know that all IP addresses in `131.159.14.0/25` are part of research unit 8 whereas `131.159.34.0/24` belong to research unit 10 of the faculty for computer science at Technische Universität München. Rather, it suffices to know that all `131.159.0.0/16` addresses are part of Technische Universität München, since they all are connected to the Internet via the same internet service provider (ISP). The longer and thus more specific prefixes for the individual institutes thus can be *aggregated* into a common shorter prefix. On the other hand, the router in the U. S. hypothetically could define an additional route for `131.159.14.0/25`, which would allow to treat packets headed for the research unit 8 differently from the other TUM traffic (e. g., in order to use a virtual network tunnel / VPN for security reasons).

ISP

aggregate

[2] Usually varying between 8 and 30 for IPv4

2.2 Routing protocols

Even with using prefixes, and even when making use of their property that they can be aggregated, the number of routes that a router is required to know is still very large in many cases (thousands and more). This inhibits manual configuration for the majority of the routes.

Instead, routers work out most of their routing information automatically by running *routing protocols*. Whenever the network topology is changed—be it due to the failure of a link, a router crashing, or components being added or removed—and thus packets to some destinations should be sent along other paths, the routing protocols ensure a prompt and automatic change in the routing. We will sketch the most important routing protocols in the following subsections.

routing protocol

2.2.1 Interdomain and intradomain routing

First due to administrative, and later due to commercial reasons, the Internet has always been divided into different areas, each of which is administered autonomously (e. g., ISPs, universities, research institutes, etc.). These areas are called *autonomous systems*, short *ASes*. The partitioning of the Internet into ASes also has great influence on the way routing is done:

AS

First, a network operator potentially has full control on the precise paths that packets travel through his own AS. However, as soon as they are handed over to a different AS, he has no further control on their path. We therefore have to distinguish between *intradomain* routing, i. e., routing of packets within a single AS, and *interdomain*, i. e., routing of packets between different ASes.

intradomain
interdomain

Second, the commercial objectives that a network operator is striving for are different in intradomain routing and interdomain routing. Within his own AS, an operator tries to configure the routing setup in such a way that the network resources (i. e., routers and links) are used in an optimal fashion. This way, the company operating the AS can relay more traffic, serve more customers and/or offer a better service quality, and thus earn more money. When it comes to data traffic leaving his AS, the objective to maximise the revenues remains unchanged; but now this goal is achieved in a different way. Most ASes earn money by selling connectivity to *customers*, while at the same time they themselves are customers of other ASes called *providers*. Obviously, network operators prefer to use network paths that maximise their revenue, while at the same time incur as little cost as possible. Therefore, if some destination d can be reached both via a provider and via a customer, then the AS in most cases prefers the route via the customer, regardless of performance considerations: This way, the AS will earn money from its customer, instead of having to pay its own provider for the traffic. Routing *policies* between ASes therefore are mostly influenced by contracts and monetary considerations, but less so by technical ones.

customer
provider

Intradomain routing thus strives to be technically optimal, whereas interdomain routing employs a so-called *policy routing*. The obvious difference in their objectives has led to the development of two classes of routing protocols that are different in scope: inter-

policy routing

domain and intradomain routing protocols. Virtually any network operator needs to cater for traffic destined for hosts within its own network, as well as traffic leaving the AS; hence many routers run an intradomain routing protocol (sometimes several ones) and an interdomain routing protocol at the same time. A router collects the routes being worked out by its different routing protocol instances in separate *routing tables* for each protocol, and combines these into its *forwarding table*, or *(general) routing table*. Potentially conflicting information is resolved by assigning each protocol a different priority, often called *administrative distance*. This way, the routing protocol with the higher priority "wins" if two or more routing protocols (and/or statically configured routes) contradict each other for a given destination.

routing table
forwarding table
administrative distance

2.2.2 BGP: The interdomain routing protocol

Today's universal de-facto standard in interdomain routing is the *Border Gateway Protocol* version 4, short *BGP* [RLH06]. BGP allows to employ routing policies in a number of ways. BGP routers exchange information on available routes in so-called *update* messages, each of which has a number of *attributes*. Whenever a router receives such an update, it checks if the update *announces* the existence of a new route, or a route that it deems to be better than the one it is has so far been using for this destination; or if a route that it is has been using until now has been *withdrawn* and is no longer available. In these cases, the receiver of the update message needs to alter its routing table, and therefore eventually itself needs to send out updates to its neighbours, informing them of the altered or withdrawn routes.

BGP
update
announcement
withdrawal

The most important attribute of a BGP update is the *AS path*. It records each individual AS that a packet needs to travel through when being sent along that route. The AS path's main purpose is to enable a loop-free[3] routing between individual ASes. BGP is therefore classified as a *path vector protocol*. Note that the AS path does not record individual routers, but entire ASes, i. e., networks.

AS path
path vector protocol

Apart from establishing a working routing setup between the ASes, the AS path together with the other attributes are intended to implement *policy routing*, i. e., they help a router decide whether a given route is favourable under its routing policies, or whether it is not.

2.2.3 Intradomain routing protocols

In contrast to interdomain routing having only one de-facto standard, there exists a multitude of intradomain protocols. Intradomain routing protocols also are referred to as *interior gateway protocols* (*IGP*), in contrast to *exterior* (=interdomain) routing protocols (*EGP*), which we described in the subsection above. We briefly sketch some of the most popular IGPs in this subsection.

IGP
EGP

As the purpose of intradomain routing is to potentially achieve a technically optimal routing, most intradomain routing protocols do not have the builtin support for policy-

[3] If there were a loop in a packet's path, then the packet would never leave this loop, until its time-to-live field reaches zero and it thus is simply discarded.

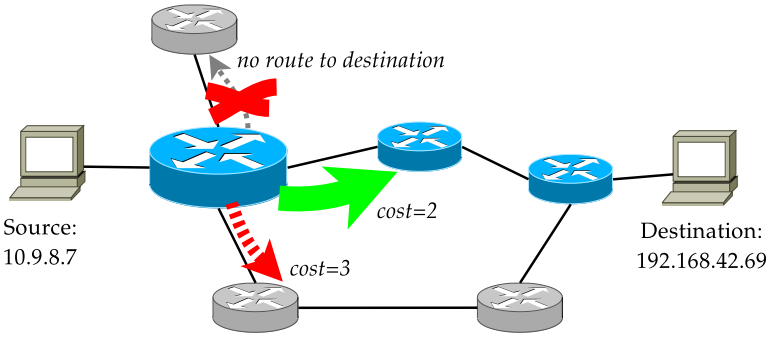


Figure 2.1: An example for a routing protocol: The highlighted router on the left knows two routes to reach the destination. To send packets, it will choose the route with lowest cost (green).

based routing as BGP has. The only notable exception to this observation is the MPLS protocol (see below), which on the other hand cannot be considered a routing protocol in the traditional sense.

IS-IS

IS-IS

In the *Intermediate System to Intermediate System* routing protocol (short *IS-IS*), each router maintains a complete view of the AS' topology. In order to ensure that all routers have the same knowledge base, the routers broadcast information on the links to their neighbours into the network at regular intervals. These broadcasts are flooded through the network. Therefore, IS-IS and similar routing protocols are termed *link-state protocols*. Additionally, each link can be assigned a *cost*, also referred to as *weight* or *distance*. Using the link-state information from their peers, every router is able to eventually construct a complete graph of the AS's topology. It then uses this graph to run some shortest-path algorithm, e. g., Dijkstra's algorithm [CLRS01], in order to calculate the best paths, thereby considering the link costs. The result of the shortest path calculation then determines the forwarding table of the router. See Figure 2.1 for a generic example.

link-state protocol
cost
weight
distance

OSPF

Very similar to IS-IS in principle, albeit more powerful (as well as more complex), is the Open Shortest Path First routing protocol, short *OSPF*. It offers better scalability, since the network can be partitioned into areas, which can help greatly reduce the number of link state broadcasts in the network.

OSPF

RIP

A simple routing protocol that is still popular with some small ISPs is *RIP*. In contrast to OSPF and IS-IS, it does not operate with link state broadcasts. Instead, each router sends information on the reachability for each of its neighbours to all of its neighbours.

RIP

Unlike in IS-IS or OSPF, this information is *not* broadcasted. Similar to BGP, the neighbours then can use this information to extend or update their routing tables to accomodate for the new routes. If the routing table of a router changes for some reason, then it announces to its neighbours all of its changed routes, together with their cost. Here, the cost of a route is the cost of the link to the next hop lying on this route, added together with the cost that had been announced in the route's update message. The cost of a route is, in the simplest case, the number of hops to the route's destination (or the corresponding egress point). The example from Figure 2.1 thus applies here as well. As the updates between routers comprise therefore a vector of (changed) routes, together with cost (=distance) information, routing protocols of this type are classified as *distance vector protocols* (*DV* protocols).

distance vector protocol

EIGRP

EIGRP

Another popular intradomain routing protocol is Cisco System's proprietary Enhanced Interior Gateway Routing Protocol (*EIGRP*), which incorporates aspects both of distance-vector and link-state protocols. EIGRP is basically a distance-vector protocol, but regarding communication with its neighbours and storing topology information, EIGRP behaves similar to a link-state protocol. Here the idea is to combine the advantages of distance-vector protocols (less signalling overhead, and computationally less expensive: the shortest paths are jointly computed by all routers, instead of being computed repeatedly and independently on each single router) together with those of link-state protocols (faster convergence to a stable state). EIGRP is therefore sometimes classified as a *balanced hybrid* routing protocol.

balanced hybrid

2.2.4 MPLS

MPLS

Multi Protocol Label Switching (*MPLS*) [RVC01] is a technique that has attracted much popularity among ISPs in the last couple of years. MPLS is not a routing protocol in the classical sense like, e. g., OSPF. The idea behind MPLS is to allow the network operator the greatest possible flexibility in choosing the paths for routing in the network, while at the same time increasing the rate of oncoming packets that a router can handle.

label

To this end, MPLS introduces a new network layer beneath the IP layer. Each IP packet that enters an MPLS-enabled network is attached an MPLS *label* (or more). Each label corresponds to a specific path through the network. Within the MPLS network, a packet is solely routed according to its label, not its IP address. As many routes share the same egress point, the operator can assign the same label to all of these routes. MPLS offers additional functionality, e. g., stacking of labels, adding, removing and exchanging labels along the path, automatically activated backup paths in the case of network failures, or support for the special needs of optical switching[4], e. g., support for different wavelengths in one glass fibre. This gives the operator a great flexibility in setting up routes, which cannot be achieved with classic IP routing that operates solely

[4] *GMPLS* (Generalized MPLS)

based on a packet's destination address. When a packet reaches an egress point and is about to leave the MPLS network, any MPLS label headers are removed, so that the next hop router sees the original IP packet that had been injected into the MPLS network.

Apart from the operator being able to configure arbitrarily sophisticated paths, MPLS offers the additional advantage of reducing the complexity of the forwarding decision, which allows to cheaper router hardware: Instead of doing a longest-prefix lookup on the destination IP address in a set of a few hundred thousand different routes at each router, this computationally expensive task (more information on this will be given in section 5.1) is performed only once, i. e., when the packet enters the MPLS network and is assigned a label. After this point, all forwarding decisions are solely made based on the packet's MPLS label(s), which can be done very fast by using simple lookup tables.

However, all the great flexibility and speed advantages of MPLS means that there is a great pice to pay: In contrast to traditional routing protocols such as OSPF, in the case of MPLS it is the network operator himself who is responsible for setting up the paths in the network, and for setting up the MPLS routers so that they have a consistent mapping from labels to paths, which are usually referred to as *label-switched paths* (*LSP*s). These LSPs usually are not set up manually, but through additional protocols such as CR-LDP [JAC+02] or RSVP-TE [BZB+97, AS03].

label-switched path

Due to this complexity, some operators combine MPLS with traditional routing protocols that are easier to handle, e. g., OSPF. As one can view the LSPs as *tunnels* through the network, these tunnels are presented as *virtual links* to an overlaying routing protocol, on which this overlaying routing protocol then can perform its shortest-paths calculations.

tunnel

virtual link

2.3 Traffic engineering

As was laid out before, the main purpose for this work is to present a method for doing dynamic *traffic engineering*, often abbreviated as *TE*. But what actually *is* traffic engineering?

traffic engineering

Due to their nature as being companies that strive to maximise their revenues, Internet Service Providers (ISPs) want to use their network infrastructure as efficiently as possible. This encompasses the task of designing the network and optimising the routing system which is responsible for delivering the traffic entering the network to its destination. In the context of IP networks, the latter task is referred to as *traffic engineering* (TE). TE techniques try to adjust the parameters of the routing system to the imposed traffic demands, in order to achieve performance gains.

2.3.1 Problem description

In its widest sense, traffic engineering is the adjustment of parameters of some network, such that the traffic conditions in the network improve. To be able to *measure* the performance of the network, some *metric* is introduced (e. g., the average travel time

measure

metric

for a packet through the network). Depending on the chosen metric, the goal of the optimisation task is either to maximise or to minimise the performance measure.

In order not to rely on trial-and-error methods in a productive network, the traffic engineer plans the adjustments in advance by modelling the problem as some form of optimisation problem. The optimisation problem is given immutable parameters (e. g., the network topology) as constants, changeable network parameters (e. g., OSPF link weights) as variables, and some goal function that captures the performance of the network, and that is dependent on the variables. To make the model useful, the goal function needs to accurately reflect performance changes in the real network when its input variables, i. e., parameters of the real network, are changed.

In the context of communication networks, traffic engineering normally describes the task of optimising the *routing* in the network: The network topology usually is considered static, since adding a link or a router is a costly investment. The traffic demands are, in most cases, considered static as well (in spite of the fact that a better network performance allows the users to take advantage of the additional capacity, thereby changing their behaviour and thus alter their traffic demands in an unforeseeable way). In this setup, it remains for the network operator to optimise routing parameters. Over time, various TE methods have been proposed. Most popular among operators are approaches that either utilise MPLS tunnels [XHBN00] or solutions with optimised IGP link weights [WXQ+06, FRT02, BLM, KKY03, FT02, FT00, FGL+00a].

2.3.2 TE metrics

Different metrics can be used to capture the performance of a communication network. In practice, many operators draw on the *maximum load of any link* in the network, and try to minimise this value. Here, the *link load*, also called *link utilisation*, is defined as the data volume per time unit sent over this link, divided by the link's capacity. The reason for choosing this metric is that heavily-loaded links suffer from network congestion, which degrades the network performance that is experienced by the end hosts—i. e., the customers of the company operating the network.

load

In addition, many network operators choose as an additional constraint to keep the load on each link below some value, typically 50 % or 70 %. Naïvely, one might assume that this threshold were an over-cautious choice, since congestion only happens if a link is utilised by more than 100 %. However, it has been shown that many traffic characteristics are consistent with self-similarity [PF95, WPT98, Bar01]. Therefore, real IP network traffic exhibits sharp peaks on small timescales which can make the link load exceed 100 % for fractions of a second—even if the average utilisation on the timescale of seconds or longer is way below 100 %. In other words, during these short packet bursts, more packets are to be transmitted via the link in question that it can actually handle. Even though very short peaks are remedied through the use of an interface *queue* that can buffer the exceeding packets until the peak is over, the router has to *drop* excess packets once the (finite-size) queue has been completely filled up. The characteristics of the widely-used [DFM+06] transport protocol *TCP* can aggravate this situation: Although a TCP transmission can easily recover from packet losses, it interprets them

queue
drop
TCP

as an indication for network congestion and therefore throttles its sending rate [Pea81]. After some time, it slowly increases its sending rate again, so that, in the end, it uses a sending rate that is well-adjusted to the capacity of the congested bottleneck link. However, with many TCP connections running in parallel over a heavily-congested link, an undesirable jojo-like bouncing effect can take place: As long as the queue of the congested interface is filled up, all packets that are subsequently scheduled to be forwarded via the interface are dropped. This packet loss usually affects many TCP connections simultaneously, each of which therefore subsequently massively throttles its sending rate. After a short period of time, the congestion is thus remedied thanks to TCP congestion control—however, since the incident affects the connections at the same point in time, they become more or less *synchronised*: Now all of them try to increase their sending rate simultaneously, which may lead to yet another congestion peak at the interface, etc. Various techniques for dropping packets in a "smarter" way have been proposed, so as to affect TCP congestion control to a lesser extent (e. g., RED), but their effectivity remains debatable [Wal03]. For these reasons, network operators are keen to minimise the link utilisation on all of their links.

synchronisation

Apart from the link load, another popular metric, especially for performance analysis of theoretical models, is *latency*, also called *delay*. When a packet travels through the network, it does not do so instantaneously, but suffers from various delays. As long as the path of a stream of packets is constant, most delays remain constant, while others are variable. Among the former are the *link delay*, which is the amount of time that it takes to encode the packet into a series of electric or optical signals that are injected into the wire (or fibre), and the *propagation delay*, which is the time that it takes to transport the data from one end of a link to another end of the link, and which obviously is determined by the physical length of the wire (or fibre) and the speed of light in the physical medium. On the other hand, the *queueing delay* is variable: It is the delay that the packet suffers at the aforementioned interface queues of the various routers that it passes through on its way. This delay obviously depends on the level of congestion in those parts of the network which the packet is travelling through.

latency
delay
link delay
propagation delay
queueing delay

2.3.3 TE in practice: Traditional TE

In their seminal paper [FT00], Fortz and Thorup present a heuristic that optimises the routing by carefully adjusting the link weights of an underlying link-state routing protocol (e. g., OSPF). The input into the network is the traffic matrix and the network topology; the output are the optimised IGP link weights. The optimisation goal is to minimise the maximum load on any link in the network. Many improvements and extensions to this methodology have been proposed since (e. g., [FT02, AC03]).

The disadvantage of these approaches is that they are quasi-static: The optimal routing is calculated offline for a predicted or previously measured traffic demand matrix. Obviously, such schemata are not able to react to unpredicted traffic changes or network events. To arrange for unforeseen sudden traffic changes, network providers account for single link failures and some traffic shifts in their optimisations [WXQ+06, FT02, AC03]. In addition, they often combine traffic engineering techniques with overprovisioning of their network infrastructure, or purely rely on overprovisioning.

2.3.4 TE in practice: TE with MPLS

One of the reasons, if not the main reason, for MPLS becoming increasingly popular with ISPs is the fine-grained and absolute control over the routing. In particular, this facilitates traffic engineering [XHBN00, AMA+99, Awd99, Kra03].

First of all, establishing well-defined paths between every pair of ingress and egress routers, a network operator can easily determine the exact traffic matrix by simply counting the number of packets entering each tunnel. This renders the sophisticated traffic matrix estimation techniques obsolete for most MPLS networks.

Second, the full control over the traffic allows the network operator to calculate the optimal routing using standard max-flow algorithms from theoretical computer science, or by solving linear optimisation problems. As MPLS allows traffic splits, these optimisation problems are not required to be formulated as integer problems and thus can be solved in acceptable computation time. The resulting optimum flows can be immediately applied to the network; there is no need to try to map from these to IGP weights.

The flexibility of MPLS does, however, come at a price, which is additional administrative complexity. This, in combination with the fact that a transition from a traditional IGP to MPLS is a radical complex step that requires extensive planning, has discouraged many network operators from applying MPLS. Recently, hybrid schemes such as, e. g., SAMTE [SBL06] have been devised that allow to use just a few MPLS tunnels while retaining the core of the network in essence under control of a traditional IGP. This approach allows to combine the advantages of IGPs (better resistance to failures, easier maintainability) and those of MPLS (greater path flexibility), and without the need to completely and radically exchange the routing setup from IGP to MPLS.

2.4 Traffic demands, traffic matrices

As was pointed out in the previous section, traffic engineering requires knowledge on the traffic demands in the network, i. e., answers to the question: how much traffic is flowing from where to where?

Reliable predictions of such network traffic demands, are, however, not only important to traffic engineering, but can be put to many uses. Internet Service Providers (*ISPs*), for example, use traffic demand matrices also for network capacity planning, to identify bottleneck links, and to evaluate scenarios in which major network assets fail. As another example, network security analysts rely on models of "normal" traffic demands to detect new threats, such as worms and distributed denial of service attacks, which generate unusual demands. As a third example, traffic demand matrices are used to drive simulators when new protocols and distributed services are designed.

Depending on their scope of the traffic, the following two types of traffic demands can be distinguished:

Intradomain traffic demands (intra-AS traffic demands) can be estimated in several ways, In principle, an ISP can read these demands directly from its routers using tools

such as NetFlow [Cla04]. The volume of data, however, introduces many complications. Furthermore, enabling NetFlow on a router has shown to result in a performance drop. This is due to the fact that router needs to update a counter that is specific to the (*source IP*, *destination IP*) tuple (or the corresponding network tuple) of the packet. Therefore, another popular approach is to use SNMP [CMPS02] to collect link-level load measurements. These impose far less computational overhead, since only one counter for each interface is needed. After collecting the link-level data of all routers at one location, it is possible to estimate a "reasonable" traffic demand matrix that is compatible with these measurements [RTZ03]. These approaches have proven effective and are used in practice today.

Modeling **interdomain traffic demands** (intra-AS traffic demands), on the other hand, is problematic because no single organisation has either the authority or the ability to measure all network traffic. An ISP can measure the demands of its clients, and the "transit" traffic that it carries on behalf of other ISPs, but even the largest Tier-1 ISP has been estimated (this is merely folklore) to carry only 7 % of the Internet's traffic. Thus, trying to derive interdomain traffic matrices is an interesting research topic.

3 Characteristics of interdomain Web traffic

In order to make routing or traffic engineering dynamic, it is important to attain a good understanding of the characteristics of the traffic that is to be routed or traffic-engineered. In particular, we need to characterise the dynamic aspects of the traffic workload. However, variability of traffic as observed in the Internet and imposed on its distributed infrastructure is determined by many factors that are not well understood.

To improve our understanding, we in this chapter introduce a methodology for estimating interdomain traffic flows on a global scale. Our methodology yields an estimation of the Web traffic between all clients worldwide and the servers belonging to over one thousand content providers. We present an interdomain traffic model that can capture changes to content, user behaviour, routing, intra- and interdomain traffic.

The methodology produces a (time-varying) interdomain HTTP traffic demand matrix pairing some hundred thousand blocks of client IP addresses with over ten thousand individual Web servers. When combined with geographical databases and routing tables, the matrix can be used to provide (partial) answers to questions such as "How do Web access patterns vary by country?", "Which autonomous systems host the most Web content?", and "How stable are Web traffic flows over time?" on a global scale.

Our methodology can estimate a significant part of the interdomain traffic demand; it can estimate HTTP traffic between over one thousand (mostly) United-States-based content providers and all clients worldwide. Our approach is based on four observations.

1. Content delivery networks (CDNs) deliver a significant fraction of the bytes downloaded by Web users. In particular, Saroiu et al. [SGD+02] observed that about 4.3 % of the Web traffic received by clients at the University of Washington between May 28th and June 6th, 2002, was delivered by Akamai [Aka].

2. For each HTTP request recorded in a CDN's Web server logs, the same client typically makes several additional requests directly to the content provider's Web servers.

3. For each object served by a CDN, the objects typically served directly by the content provider can be identified by examining traces of Web usage from large groups of users, or by examination of the content provider's Web site.

4. The locations of the content provider's Web servers can be determined with the help of the DNS system and information available from the interdomain routing system.

In this chapter, we combine server logs from Akamai's CDN network with HTTP traces extracted from packet traces gathered at several universities to build detailed traffic demand matrices. We provide two types of matrices: *Publisher demand* matrices pair hundreds of thousands of client IP blocks with over one thousand publishers. *Web traffic demand* matrices pair these client blocks with tens of thousands of IP addresses belonging to publisher and CDN Web servers. For each pair, in either type of matrix, we estimate the rate at which data is transferred to the clients in the block.

Some caveats

Our traffic matrices are among the largest, in terms of the number of pairs, ever generated, and much of the traffic crosses domain boundaries. Nevertheless, there are a number of limitations to our work. First, we are capturing only one class of traffic, HTTP. Although several studies have shown that HTTP traffic is among the most common, its dominance has recently been challenged by new classes of traffic such as peer-to-peer file sharing data and streaming media. Whether or not HTTP traffic demand can be used to effectively infer overall traffic demand is an open question. Second, we are not capturing the HTTP traffic sent by all Web servers, but only those belonging to Akamai and its CDN-customers. Even though many of the most popular Web sites utilise Akamai's servers, there are many thousands that do not. Akamai's customer base is also biased towards U.S.-based companies. Third, we assume that the number of bytes served by the content provider for each Akamai object can be estimated by examining traces from a small number of large client sets. In practice, content providers might tailor their Web pages for different client sets. For example, a U.S.-based site might choose to serve more compact (fewer bytes) Web pages to overseas clients.

3.1 Related Work

The book by Krishnamurthy and Rexford [KR01] contains an excellent survey of Web usage studies. Some of the work most closely related to the method described in this chapter has focused on understanding either user behaviour [CB97, WPT98, AW97, ISZ99], some aspects of changes in content, [WM00], on the effects of these changes, e. g., in terms of the traffic demands [Awd02] imposed on a tier-one ISP [FGL+00b, MTS+02, MFT+02, XHBN00], or in terms of poor end-to-end performance experienced by the users. The latter can be observed via active measurements of delay, loss, or throughput [PAMM98], or passive monitoring of individual routers and links [TMW97].

As mentioned above, there are a variety of approaches for estimating intradomain traffic matrices. Indeed, this topic has been the subject of intense research over the past years [RGK+01, ZRLD03, ZRDG03, MFT+02, MTS+02, SNL+04, LY03, FGL+00b].

Over the same period of time, traffic engineering has grown greatly in importance, and a large effort has been devoted to designing intradomain traffic engineering solutions (see Section 6.1). This timing is not coincidentally—one of the primary inputs to most "traffic engineering" algorithms is an estimated traffic demand matrix (see Section 2.4).

Even more recently, a number of schemes for Interdomain traffic engineering have been proposed [FBR03, Qea03, QUB02, UBQ03, ACK03, WJR02, QYZS03]. Yet, not much work has been published on estimating interdomain traffic demands on a global scale. Even the question of whether interdomain traffic matrices and intradomain traffic matrices have similar dynamics is unanswered. *Partial* interdomain traffic matrices have been made public to researchers [UQLB06]. Recently, Chang et al. have provided an approach to modelling inter-AS traffic matrices from publicly available data, as well as from various and extensive Internet-wide measurements, including search engine queries [CJMW05]. In Chapter 4, we present a methodology for studying the behaviour of users performing Web search sessions. The results of our analysis can be used to refine the methodology of Chang et al. .

3.2 Background: CDNs and Terminology

Before we describe our methodology in further detail, we provide the reader with a brief overview of the process of content delivery with and without content delivery networks (CDNs). We also present a brief dictionary of the terms and abbreviations used in the remainder of the chapter.

3.2.1 Terminology

The following definitions, taken in part taken from the Web Characterization Terminology & Definitions Sheet [LN], will serve to clarify the subsequent discussions.

Web site: A collection of interlinked Web objects hosted at the same network location by a set of **origin Web servers**. — **Web site** **origin Web server**

Web site publisher or just **publisher**: A person or corporate body that is the primary claimant to the rewards or benefits resulting from usage of the content of a Web site. A publisher may distribute his content across multiple Web sites. Publishers are also referred to as **content providers**. — **publisher** **content provider**

Content delivery network (short **CDN**): An alternative infrastructure operated by an independent service provider, on which some parts of a Web site can be hosted. — **CDN**

3.2.2 Content delivery

The purpose of the Internet is to allow the exchange of and access to information. This information is typically hosted on *origin Web servers*. Content delivery networks, short CDNs (see, e. g., [Hul02, DMP+02, GCR01, BV02, JCDK00, KWZ01, SGD+02]) are designed to reduce the load on origin servers and at the same time improve performance

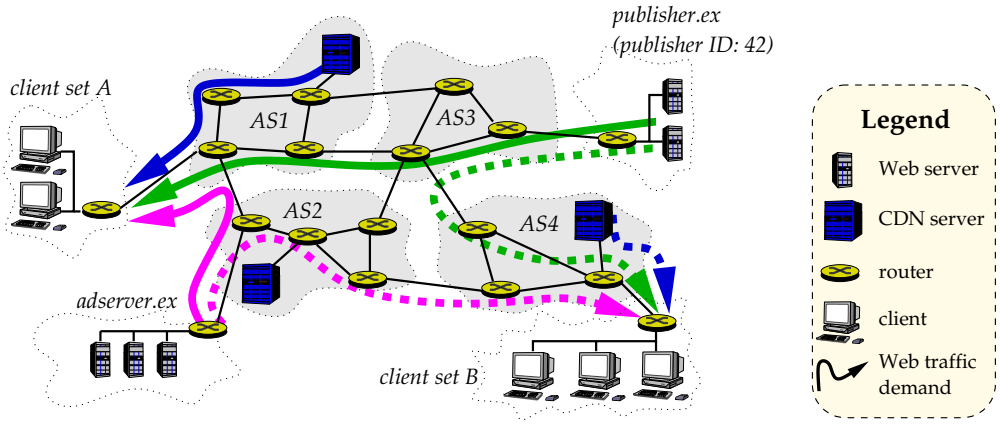


Figure 3.1: Example of CDN deployment and traffic flows (Web traffic demands).

for the user. Most CDNs have a large set of servers deployed throughout the Internet and cache the content of the original publisher at these servers. Therefore another view of CDNs is that they provide reverse proxy services for content providers, the publishers. In order to take advantage of their distributed infrastructure, requests for data are redirected to the "closest" cache server. Intelligent redirection can reduce network latency and load (and therefore network congestion), thereby improving response time. CDNs differ in their approach to redirecting traffic. Some, such as Akamai [Aka], use DNS to translate the hostname of a page request into the IP address of an appropriate nearby CDN server. This translation may consider the location of the client, the location of the server, the connectivity of the client to the server, the load on the server, and other performance and cost based criteria.

An example that shows how the CDN infrastructure is embedded in the Internet architecture is shown in Figure 3.1. We can see four ASes, numbered 1–4, two Web site publishers, `publisher.ex` and `adserver.ex`, and two sets of clients. The publisher `publisher.ex` is connected to AS 3; the publisher `adserver.ex` is connected to AS 2. A set of clients is connected to AS 1, another one to AS 4. Traffic is routed between the ASes by means of BGP (or some other exterior gateway protocol); traffic within an AS is routed by means of Interior Gateway Protocols [Hal97].

The location of the CDN's servers differs from CDN to CDN and depends on contractual agreements between the CDN and the individual ISPs. In some instances, the CDN servers are deployed within the data centers of the ISP and therefore belong to the same AS, like AS 1, 2, 4 in Figure 3.1. Clients of the ISP (i. e., directly connected end users) will typically be served by these servers in the same AS. With other ISPs, the CDN may have a private peering agreement that allows the CDN to serve requests from the ISP's clients via a direct connection between the CDN and the AS. The CDN may also co-locate servers with the ISP's clients, e. g., on university campuses. Other ISPs may have no relationship with the CDN, and the traffic to the ISP's clients is thus routed via another AS.

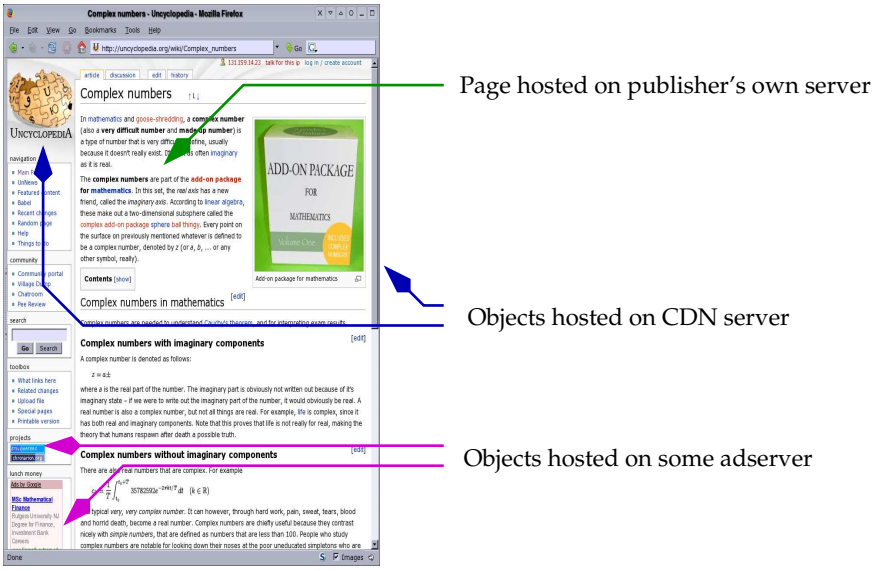


Figure 3.2: Example Web page with some CDN content and some content hosted on an Web advertisement server.

Let us now consider the steps that are necessary to download the Web page shown in Figure 3.2. This page consists not only of the main HTML page, which is located at `publisher.ex/page.html`, but also several embedded objects.[1] The publisher responsible for `publisher.ex` has decided to use the services of a CDN, `cdn.ex`. Therefore, two objects (the two graphics in the top section) of the sample page are hosted by the CDN. Moreover, an image and an internal frame are provided by another company providing dynamic advertisements, `adserver.ex`.

Whenever a specific client from client set A in Figure 3.1 on the previous page accesses the Web page, `publisher.ex` will serve the bytes for the main page and one embedded object, whereas `adserver.ex` will serve the bytes for the advertisements, and the "nearest" CDN server will serve the two CDN-located objects—in our case, they will be served from AS 1. In contrast, if instead a specific client from client set B accesses the page, the two CDN objects will be delivered from a different CDN server, namely the one in AS 4. Keep in mind that it is the objective of the CDN to direct the client to a CDN server that is close to the client.

3.3 Interdomain Web traffic demands

In this section we motivate and introduce abstractions for publisher demands and Web traffic demands and discuss some possible applications based on these abstractions.

The interplay between content hosting, intra- and interdomain routing, and the Internet architecture affects the set of traffic demands we choose to estimate. In contrast

[1] For simplicity reasons, we assume that the page consists of one single HTML file, i. e., it does not comprise any additional files, such as external style sheets, Javascript files, etc.

to previous work, we are not focusing on a single ISP. Rather, the goal of this study is interdomain traffic imposed by *any* client through accessing content provided by *many* publishers, on a global scale. The situation lends itself to two abstractions:

publisher demand matrix

1. a *publisher demand matrix* that captures traffic behaviour at the aggregate level of a publisher or content provider; it pairs each client IP block with various publishers

   and

Web traffic demand matrix

2. a *Web traffic demand matrix* that captures the traffic at the granularity of a Web server with a specific IP address; it pairs each client IP block with various Web server IP addresses.

3.3.1 Motivation

Traffic demands usually specify the amount of traffic flowing between two end-points, from the source to the destination, which is sufficient as long as both end-points are of the same granularity. In the context of Web traffic, treating end-points at the same granularity is problematic, as there are many more clients than servers or publishers. Distinguishing between individual clients is moot due to the sheer size of the resulting matrix.

Just as the interplay between intra- and interdomain routing motivated a point-to-multipoint demand model [FGL+00b], it motivates us to define Web demands in terms of network prefixes that are consistent with BGP. This enables us to address questions arising in the context of inter- and intra-domain routing as well as questions regarding how to multi-home sites and how to balance traffic between ISPs.

Summarising clients according to network prefixes appears appropriate. Network prefixes provide a way of aggregating client traffic that preserves locality in terms of the Internet architecture. Such an aggregation is necessary in order to avoid the severe scalability problems of representing each client at the level of an IP address. In addition, it reduces the statistical significance problem caused by too little traffic per individual IP address.

Yet, summarising publishers via network prefixes is hazardous. A publisher that serves gigabits per second to clients is likely to use a distributed infrastructure together with some load distribution mechanism, such as DNS round-robin or proximity-aware routing. In general, these mechanisms are very similar to those employed by CDNs. This usually means that the content is available via multiple IPs in different network prefixes. Furthermore, it is sometimes impossible to deduce the Web site publisher from its IP address: One HTTP server may host multiple sites of several publishers. Even the URL of an object does not directly allow us to infer the identity of the publisher for the content, e. g., to see that Vivendi Universal Interactive Publishing is responsible for `www.lordoftherings.com`. Some publishers split their content into various sites, each with its own responsible organisation and its own independent infrastructure. All these issues imply that one may want to capture the traffic matrix at two levels of abstractions: at the publisher level or at the level of each individual Web server.

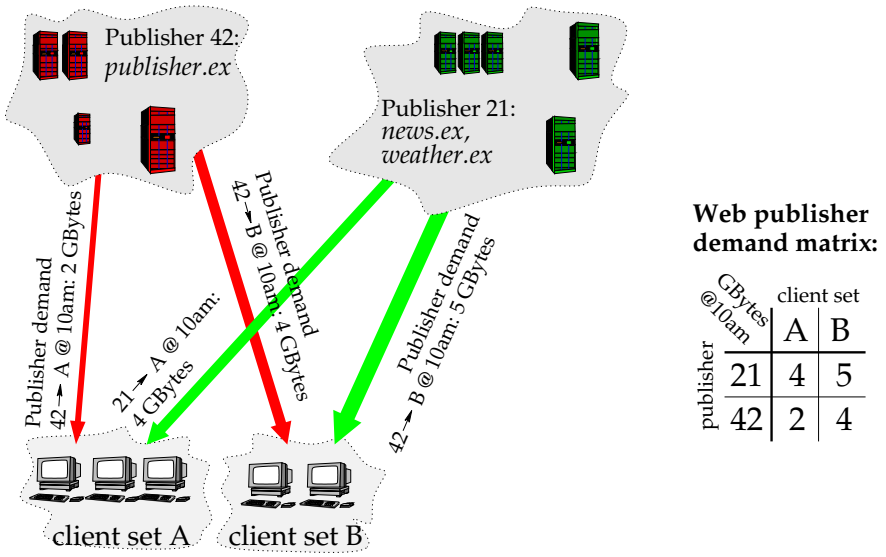


Figure 3.3: Publisher demands.

3.3.2 Examples

Having motivated the need for the two kinds of matrices—publisher demand and Web traffic demand—, we now present some illustrative examples. Figure 3.3 shows two different publishers that are identified by ID numbers 42 and 21, and the domain names of the sites that they publish: `publisher.ex` for 42, `news.ex` and `weather.ex` for 21. Their content is accessed by two different client sets: A and B. Each client set accesses some of the content provided by `publisher.ex` and `news.ex/weather.ex`. This results in traffic flowing from the Web sites of `publisher.ex` and `news.ex/weather.ex` to the client sets A and B. These traffic flows are what we refer to as *publisher demands*.

publisher demand

If we want to improve, say, our routing decisions, then publisher demands are not of much use: They do not take into account the server locations. In the distributed infrastructure for the publisher with ID 42 shown in Figure 3.1 on page 34, some of 42's content (namely `publisher.ex`) is hosted at servers connected directly to AS 3, some of 42's content has been offloaded to a CDN; furthermore there may be third-party content such as banner ads hosted by `adserver.ex` on some of 42's pages. In Figure 3.1, the resulting three *Web traffic demands* to client set A are indicated by the smooth arrows; the Web traffic demands to client set B are depicted by the dotted arrows.

Web traffic demand

3.3.3 Applications

These notions of demands enable experimentation with changes to content hosting, to routing, to the AS level topology, as well as to the location of the content and/or the clients. A publisher that needs to upgrade its infrastructure has many choices: upgrade the existing servers, add more servers, add more bandwidth to existing network con-

nections, add alternative network connections, change the way requests are allocated to individual servers, or outsource more of its content delivery. In order to decide on the best option, the publisher may use the publisher demands to evaluate possible scenarios: the traffic volume imposed by different client sets may influence his decisions. For such "what if" scenarios, he needs to understand the dynamics of both the publisher demands, as well as the Web traffic demands, as well as the differences in the dynamics between them.

An ISP may also need to predict the effects that adding or moving a link or peering session may have. This requires a model of interdomain traffic. An important difference between traffic statistics collected within an AS and the Web traffic demands discussed here is that they describe traffic flows not just through the network of the ISP, but throughout the Internet. Therefore, given an understanding of the dynamics of Web traffic demands, it is easier to estimate the effects that decisions (such as adding peering connections) may have. Furthermore, it is possible to explore what effects policy changes will have. Applying our methodology, this is feasible not just for policy changes by the ISP itself, but also for policy changes performed by other ISPs.

By combining Web traffic demands with topology and BGP routing information, one can explore the impact of routing instabilities on actual traffic flows and vice versa. Moreover, by combining the Web traffic demands with performance measurements, one can explore how user feedback should be factored into future decisions. Furthermore both demands, the Web traffic demand as well as the publisher demand, are ideal inputs for driving interdomain network simulations.

3.4 Using CDNs to estimate publisher demands

The publisher demands can be computed in two ways—either given information from each publisher regarding which clients access its content and from which prefixes, or given information from each client set about which Web sites they are requesting. One way of deriving this information would be to collect fine-grain traffic measurements at all publisher sites or all client sites. This may enable us to identify the traffic as it reaches the Web site publisher or the clients. However, this approach is virtually impossible, since the huge number of publishers and client sets imposes a task that is simply unmanageable. Furthermore, it would still be necessary to address the question of how to distinguish publishers that are co-located at the same server. Just analysing a large proxy log does not help either, since it does not allow us to gather information about any significant subset of all possible clients.

Instead, we focus on publishers, as there are far fewer publishers than clients. Yet, instead of considering all publishers, we take advantage of the fact that CDNs provide reverse proxy services (Section 3.2.2) for the content providers (the publishers); they thus are acting as "subcontractors" to the publishers. Using data collected within CDNs has several advantages:

- CDNs serve the content on behalf of their customers (the publishers). This implies that the CDN has a way of relating content to publishers.

- Due to the requirements imposed by volume-based billing, CDNs collect data on behalf of the publishers regarding how much traffic is served. This implies that the CDN has a way of deducing the amount of traffic it serves on behalf of each individual publisher.

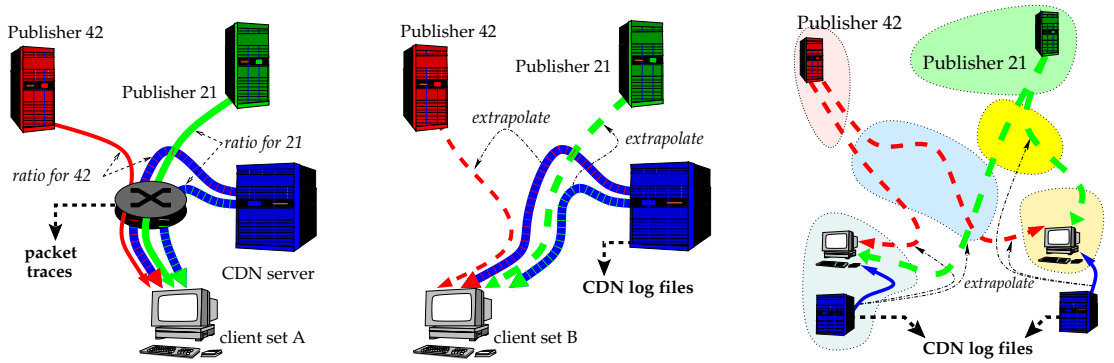
- In addition, most publishers do not want to lose access to the information they can collect when they serve content directly to clients. For example, information about which clients are accessing what content is derivable from Web server logs. Accordingly the CDN has to collect this "Web server"-like log information. As a consequence, it has a way of relating traffic to clients.

Moreover, the number of CDN service providers is significantly smaller than the number of publishers. A list of CDN types and their products is maintained by Davison [Dav03]. To further reduce the candidate set, we observe that, at the time of this writing, the market is dominated by only a small number of CDNs such as Akamai [Aka], Speedera, Cable & Wireless and Mirror Image [Mir].

Focusing on CDNs limits us in terms of the number and kind of publisher demands that can be estimated: If a publisher has no association with a CDN, it will not be possible to derive his publisher demands. This raises the question of which publisher demands we are interested in, and if those are likely to be associated with a CDN. Like a lot of other quantities in networking [BCF+99, FP99, FGL+00b] and elsewhere [Zip], we expect publisher demands to be consistent with a Zipf-like (cf. glossary) distribution. Since the heavy hitters account for a significant part of the traffic, we are mainly interested in them. Luckily, those are the ones that are more likely to use the services of a CDN. Therefore CDNs can provide us with a way of estimating the publisher demands for those content providers that are most popular and thus account for a large part of the traffic.

Still one problem remains: As discussed in Section 3.2.2 and as shown in Figure 3.1, CDNs try to take advantage of their distributed infrastructure by serving traffic locally. So how can we expect to derive estimates for interdomain Web traffic demands from CDN traffic data? Here it turns out that most publishers will not serve their whole content via the CDN. Rather, they will use some mixture, as shown in Figure 3.2. Note that not all content has to be served via the Web site of the publisher or the CDN—some embedded objects may be located on yet another server instead, e. g., banner advertisements.

Together this provides us with the opportunity that we need: If we know the ratio of a customer's traffic serviced via a CDN vs. the traffic from the publisher's own servers vs. traffic served by via external sites (see Figure 3.4(a)), and if we know the traffic serviced by the CDN (Figure 3.4(b)), we can estimate the other amounts (Figure 3.4(c)). These facts allow us to estimate publisher and Web traffic demands for *all* client prefixes world-wide and *all* publishers that are customers of the CDN. Our methodology significantly improves the availability of interdomain traffic estimation on a global scale—so far at best a scarce quantity.



(a) Determining traffic ratio $\frac{\text{publisher traffic}}{\text{CDN traffic}}$

(b) Estimating unseen non-CDN traffic

(c) Resulting interdomain traffic demand matrix

Figure 3.4: Web publisher demand estimation.

3.5 Realisation ideas

With access to the logs of a CDN, determining the traffic served by the CDN on behalf of a specific publisher is possible. Accordingly, we now discuss how we approach the remaining problems of how to estimate traffic ratios between publisher and CDN traffic, as well as how to map publisher demands to Web traffic demands. Further details are provided in Section 3.6.

3.5.1 Estimating traffic ratios

One way to proceed is to explore the content provided by the Web site of the publisher offline, or by using Web crawlers. Given a set of Web pages, one can easily calculate the fractions of data served by the CDN vs. the fraction of data served by the original Web site. The problem with this approach is that it ignores the fact that certain Web pages are more popular than others.

Hence, we really need access to information about actual user accesses. There are many ways of doing this [KR01]: from users running modified browsers [CB97], from the logs of the publishers themselves [AW97], from proxies logging information about which data is requested by the users of the proxy [PM95, KLM97], or from the wire using packet monitoring [Fel00a, BPS+98, GB97]. Each of these methods has its advantages, but most have severe limitations regarding the detail of information that they log. Distributing modified Web browsers suffers from access to the browser software and from users not accepting the modified browsers. Although a few publishers actually might cooperate by revealing their logs, most will not. In addition, this approach suffers from a scalability problem. Using proxy logs or logs derived via packet monitoring may be more scalable with regards to ISPs—but with regards to the size of the user population that can be monitored, it is more limited.

To choose the appropriate solution, let us consider the granularity at which we need the information. The purpose of estimating the publisher demands is mainly to understand their medium time-scale fluctuations and their impact on traffic engineering, routing, etc. We are not as interested in small time-scale events (and in any case it is hard to understand their causes). Therefore, some coarse-grain estimation is sufficient for our purposes. Hence, we propose the following two-fold approach:

- Obtain from the publisher their estimate of the fraction of traffic that is served by the CDN and other third party providers; admittedly, we utilise the provider-customer relationship between the CDN and the publisher to acquire this information, which is provided by only a subset of the publishers.
- Use packet level traces or proxy logs to derive the fractions for some users and therefore for some sample client sets. (Although proxy logs suffice, since detailed timing information is not required, the analysis in this chapter is based on packet level traces.)

Consider again the example page shown in Figure 3.2 on page 35. A log file, derived from a proxy log or the packet traces, should show five entries per access to this page, i. e., one for the HTML file of the page, and one for each embedded object (unless it is cached in the user's cache). Each entry includes an `object_id` (i. e., the URL), the start and end time of the download of the object, the number of transferred bytes, and the `HTTP_REFERRER` [2] field (if specified by the user agent). Note that the referrer field, which lets a user agent include the URL of the resource from which the requested object was reached, is optional and not necessary. Nevertheless, most popular Web clients, such as Firefox and Internet Explorer, include them regularly, which proves to be extremely helpful. In our sample page, all embedded objects have the same value for their referrer field independent of where the object actually resides: The value is the same as the URL of the base page. Thus the referrer field provides us with the means to associate the objects and therefore provides us with the means of estimating the ratios between the traffic flows.

One way of estimating the ratios could be to try to compute the exact temporal and causal relationship between the pages and their embedded objects. But past work, e. g., in the context of estimating the benefits of prefetching [KLM97] or piggybacked cache validation [KR01], has shown that this is a nontrivial task, especially in the presence of proxies and strange user behaviour. For our purpose, the fact that there *is* a relationship is sufficient. See Section 3.6.2 for further details.

3.5.2 From publisher demands to Web traffic demands

In order to derive the Web traffic demands from the publisher demands, we first need to map the Web sites of the publishers to IP addresses. This mapping may not be a one-to-one mapping—recall that some publishers use a distributed infrastructure and

[2] The header field `HTTP_REFERRER` is misspelled in the official protocol definition of HTTP [FGM+99]. Unless we explicitly refer `:-)` to the HTTP header field, we will use the correct spelling referrer.

therefore apply DNS mechanisms for "load balancing", "proximity-aware", or "server-feedback dependent" name resolution, in a manner similar to Akamai's mechanism for distributing load, or even entrusting Akamai to provide these mechanisms.

Again, we propose to take advantage of information available to the CDN: It knows the set of hostnames that are associated with each publisher. Therefore the problem is reduced to associating each hostname with its set of IP addresses.

This can be done using DNS queries. To account for "proximity-aware" or "server-feedback dependent" policies used by the publisher, it is not sufficient to issue DNS queries from a single point in the Internet—rather, we need to use a set of DNS servers that are distributed throughout the Internet. Since we have to issue recursive queries[3] to these servers in order to discover their view of the server IP addresses, they have to allow recursive DNS queries.

In a second step, we determine which server is used by which client. This problem can either be extremely simple or extremely hard. If the site uses a single IP address or simple DNS round robin across a number of different IP addresses, this step is trivial. Since DNS round robin is supposed to partition the requests about evenly across all of the servers, this is what we will do in estimating demand. If, however, the site uses a more sophisticated mechanism, we are left with a fairly difficult problem. Here we have two possible ways to approximate the decision of the physical Web site: We can either use the result of the *DNS* server "closest" to the client set, or we can assume that the client set is directed to the "closest" *Web* server. As we are dealing with an estimation of inter-AS traffic in this chapter, we propose to capture the meaning of "close" in terms of AS distance. This seems reasonable, since it is known that some distributed infrastructures are using this information [Liu], and since other measures of closeness are even harder to define.

3.6 Implementation details

In this section we present more details of the ideas outlined in the previous section. More specifically, we describe how we estimate publisher demands and Web traffic demands using logs from a CDN provider, packet level measurements at ingress links, and the DNS system.

3.6.1 CDN log evaluation

To compute publisher demands using CDNs, fine-grain access records from all servers of the CDN have to be collected. Usually, servers generate a record summarising each transaction. These records are exported on a regular basis for billing purposes and include sufficient information for computing the publisher demand: the `accessed_object`, the `client` IP address, the `start` and `end` times of the transfer, and the

[3] In an iterative query, the contacted name server tells the requesting name server which name server to ask next, whereas in a recursive query the contacted name server proceeds by sending a query to the next name server on behalf of the original user.

number of `transferred_bytes`. (Any additional information can be used to further refine the notion of publisher demands.)

Computing the traffic demands requires information about the CDN customer (i. e., publisher) associated with each record. This aggregation process draws on a map, `object_to_customerid`, such that every `object` can be associated with a unique `customer_id`. Furthermore, it uses another map, `clientip_to_client_prefix`, of network addresses such that every source IP address (`client`) can be associated with a network prefix `client_prefix`. The first map can be derived from the customer information of the CDN while the second can be derived with longest prefix match from a joined BGP routing table (`joined_bgp_table`) from multiple different viewpoints in the Internet. Alternatively, one can use static groups such as up to the /24 level, which (given that most ISP will not allow propagation of prefixes smaller than /19) does not hinder any later application specific aggregation.

No content transfer is instantaneous. Rather, they last for some time interval starting at `start`, ending at `end`, and contributing some amount of traffic, `transferred_bytes`. In order to avoid problems in time resolution, e. g., discrepancies between clocks at the record collectors, granularity of the data sources, etc., and since most applications making use of publisher demands are on a larger time scale, we compute the demands on time scales of multiples of minutes rather than seconds. Time is partitioned in bins of duration `bin_length`, according to the considered resolution. If a record spans multiple bins, we subdivide the traffic in proportion to the fraction of time spent in each time period.

To derive the final publisher demands, we draw on another map, `customerid_to_demand`. It specifies for each `customer_id` the relationship between the CDN-hosted traffic flows and the self-hosted traffic and is the result of the computation detailed in Section 3.6.2. The mechanism for computing the publisher demands is summarised in algorithm 1 on the next page.

3.6.2 Estimating flow ratios between CDN and publisher

In Section 3.5, we suggest using proxy and/or packet level traces to estimate the relationships between the various flows shown in Figure 3.4(b) on page 40. We now present a three-pass approach which automatically ensures that Web pages referring to other Web pages are handled appropriately.

The first two passes serve preparative purposes. In the first pass, we separate the set of accessed objects according to the client IP addresses. In the second pass (algorithm 2), we determine the set of objects served by the CDN under consideration, `cdn_set`, and some additional information that we specify below. For this purpose, we check each object against the appropriate CDN customer base information (`determine_customer_id()`) and, if appropriate, compute the CDN `customer_id` and add it to the `cdn_set`.

In the third pass, we compute for each CDN object `cdn_id` within this set the feasible base pages `base_candidate_set` and the set of possible other embedded objects `embedded_candidate_set`. For an object to fall into these sets, either its URL or its referrer has to be equal to the referrer value of the CDN object. For this pur-

Algorithm 1 Estimating CDN publisher demands from CDN transaction logs.

for all *accessed_object*: (*client*, *start*, *end*, *transferred_bytes*) **do**
  customer_id ← object_to_customerid(*accessed_object*)
  client_prefix ← longest_prefix_match(*client*, *joined_bgp_table*)
  $start\_bin \leftarrow \lfloor \frac{start}{bin\_length} \rfloor \cdot bin\_length$
  $end\_bin \leftarrow \lfloor \frac{end}{bin\_length} \rfloor \cdot bin\_length$
  if *start_bin* == *end_bin* **then**
    volume[*client_prefix*, *customer_id*, *start_bin*] += *transferred_bytes*
  else
    ▷Compute volume of traffic for each *time_bin*:
    $byte\_rate \leftarrow \frac{transferred\_bytes}{end - start}$
    volume[*client_prefix*, *customer_id*, *start_bin*]
      += $byte\_rate \cdot (start\_bin + bin\_length - start)$
    for $time\_bin \leftarrow start\_bin + bin\_length$; $time\_bin < end\_bin$;
      $time\_bin$ += *bin_length* **do**
      volume[*client_prefix*, *customer_id*, *start_bin*] += $byte\_rate \cdot width$
    end for
    volume[*client_prefix*, *customer_id*, *end_bin*] += $byte\_rate \cdot (end - end\_bin)$
  end if
end for
for all aggregate **do**
  demand[*client_prefix, customer_id, end_bin*] ← *customerid_to_demand*[*customer_id*] · *volume*[*client_prefix, customer_id, end_bin*]
end for
Output for each aggregate: (*client_prefix*, *customer_id*, *time_bin*, *volume*)

pose, we have stored some additional information in the second pass: each object with URL `url` and referrer `referrer` is added to the set of possible home pages for this URL `base_set(url)`. Furthermore, we add the object to the set of possible embedded objects for the current referrer, denoted by `embedded_set(referrer)`. Once we have retrieved the candidate sets, we can determine the hostnames for each of the objects within the candidate sets, and add the bytes in the corresponding object to the appropriate traffic flow. The appropriate traffic flow is either determined by the `cdn_customer_id` for CDN objects or the hostname for non-CDN objects. If the hostname is not used in the client's request, we propose to use the server IP address instead. In order to keep the relationship information, we can now establish the link `associated_hosts` between `cdn_customer_id` and the hostname of the objects in the candidate sets. In order to avoid double counting, e. g., if the exact same page is accessed multiple times, one needs to mark every object that has already been accounted for.

Again, it is the case that no content transfer is instantaneous—but rather than spreading the contribution of each transfer across multiple time periods of duration `bin_`

Algorithm 2 Computing flow ratios: CDN server accesses vs. publisher server accesses from user access logs.

1: ▷Pass 1:
2: Sort the accessed objects according to client IP addresses
3: ▷Pass 2:
4: **for all** *client_IP* and *object_id*: (url, start, end, trans_bytes, referrer, hostname) **do**
5: **if** determine_customer_id(*object_id*) evaluates to a CDN object **then**
6: *customer_id*[*object_id*] ← determine_customer_id(*object_id*)
7: *cdn_set* ∪= *object_id*
8: **end if**
9: *base_candidate_set*[*url*] ∪= *object_id*
10: *embedded_candidate_set*[*url*] ∪= *object_id*
11: **end for**
12: ▷Pass 3:
13: **for all** *object_id* ∈ *cdn_set* with (url, start, end, transferred_bytes, referrer, hostname) **do**
14: **if** *done*[*object_id*] **then**
15: **next**
16: **end if**
17: *done*[*object_id*] ← true
18: $end\_bin\_cdn \leftarrow \lfloor \frac{end}{bin\_length} \rfloor \cdot bin\_length$
19: *cdn_customer_id* ← *customer_id*[*object_id*]
20: *volume*[*cdn_customer_id*, *end_bin_cdn*] ∪= *transferred_bytes*
21: **for all** *candidate* ∈ *base_candidate_set*[*referrer*] ∪ *embedded_candidate_set*[*referrer*] **do**
22: **if** ∃ *customer_id*[*candidate*] or *done*[*candidate*] **then**
23: **next**
24: **end if**
25: *done*[*candidate*] ← true
26: *associated_hosts*[*cdn_customer_id*] ∪= *hostname*[*candidate*]
27: $end\_bin\_candidate \leftarrow \lfloor \frac{end[candidate]}{bin\_length} \rfloor \cdot bin\_length$
28: *volume_related*[*cdn_customer_id*, *hostname*[*candidate*], *end_bin_candidate*] ∪= *trans_bytes*
29: **end for**
30: **end for**
31: Output for each *customer_id* and *host* from the *associated_hosts* the ratios:
(*customer_id*, *hostname*, *time_bin*,
volume[*customer_id*, *time_bin*], $\frac{volume\_related[host,time\_bin]}{volume[customer\_id,time\_bin]}$)

length, we propose to just add it to the last bin. It is known from aggregating NetFlow data that this can lead to artefacts [SF02]. But if the aggregation periods are long enough, size and impact of these artefacts decrease significantly.

3.6.3 Mapping publisher demands to Web traffic demands

In order to map the publisher demands to Web traffic demands, we need to find out which IP addresses are actually in use by the publisher's infrastructure. As an initial step, we derive the set of hostnames associated with each site publisher (customer_id; via the mapping customerid_to_hostname), utilising the knowledge of the CDN provider. Therefore, the problem is reduced to associating each hostname (host) with its set of IP addresses (ip_set).

To account for the distributed infrastructure of the site, we have to issue recursive DNS queries from a set of DNS servers that are distributed throughout the Internet. We propose to identify a set of candidate DNS servers from traffic measurements, such as NetFlow or packet level traces, or by checking the server logs of the CDN's DNS server. Using packet traces has the advantage that its easy to check if the DNS servers support recursive DNS queries. Otherwise, one can issue a recursive query to the DNS server and see if it is willing to respond to the query, and second, if it does support recursive queries. Once we have derived a candidate set of DNS servers one way or the other, we can either use all of them, or some subset. We propose to concentrate on a subset, such that each DNS server in the subset will return a different IP address for at least one Web site publisher that utilises a distributed infrastructure. Since the CDN runs a highly distributed infrastructure, we propose to use the main Web server of the CDN for this purpose, in our case akamai.com and www.akamai.com.

The next step involves identifying what kind of access distribution mechanism (dns_policy) is used by the physical Web site. We propose to concentrate on popular mechanisms and look for indications of their use (algorithm 3). If all queried DNS servers return almost the same set of IP addresses, then we can assume that DNS round robin ("round robin") is used. We use "almost" instead of "exactly", since one cannot query all DNS servers at the same time; this lack of synchrony can cause anomalies. If, however, different DNS servers return different IP addresses in a consistent fashion (at least twice), then we can assume that some form of proximity-aware load balancing is used ("proximity").

In the first case, we propose to split the load evenly between all IP addresses that are used to implement the physical infrastructure. Otherwise, we propose to split the traffic only between those IP addresses that are resolved by the "closest" (Section 3.2.2 on page 34) DNS server queried from the clients in question. All other cases are currently resolved via manual inspection.

Algorithm 3 Mapping site publishers to Web traffic demands.

```
1:  for all customer_id do
2:    hostname_set ← customerid_to_hostname(customer_id)
3:    for all host ∈ hostname_set do
4:      for all dns_server ∈ dns_server_set do
5:        ip_set[customer_id] ∪= dns_query(dns_server, host)
6:        ip_set_dns[customer_id, dns_server] ∪= dns_query(dns_server, host)
7:        dns_policy[customer_id] ← classify_dns_policy(ip_set)
8:      end for
9:    end for
10: end for
11: for all client_prefix do
12:   closest_dns_server[client_prefix] ← closest(client_prefix, dns_server_set)
13: end for
14: for all customer_id and client_prefix do
15:   if dns_policy[customer_id] is "round robin" then
16:     split traffic evenly among ip_set[customer_id]
17:   else
18:     split traffic evenly among
                  ip_set_dns[customer_id, closest_dns_server[client_prefix] ]
19:   end if
20: end for
```

Dataset	Location	Key Fields
CDN sites	CDN	List of Web sites and publishers that use the CDN
CDN servers	CDN	List of hostnames of physical Web sites
CDN logs	CDN's billing system	Per accessed object: client IP address, resource, start time, end time, transferred bytes
HTTP logs	external network connection	Per accessed object: client IP address, URL, start time, end time, transferred bytes, referrer, hostname
DNS lookups	set of name servers	Per hostname and DNS server: set of IP addresses
BGP table	peering points	Per network: set of possible routes (AS path)
EdgeScape	CDN	Maps IP addresses to locations

Table 3.1: Datasets and key fields used in computing and validating the publisher and content traffic demands.

Dataset	Date	Duration	Size
CDN logs	{Apr 26th, Apr 28th, May 5th}, 2004	3×2 hrs	617.4 GB `.gz`
HTTP logs	Jan 30th, 2004 – May 11th, 2004	102 days	28.5 GB `.gz`
DNS lookups	May 12th, 2004 – May 13th, 2004	1 day	5.4M queries
BGP tables	Apr 28th, 2004	—	270 tables

Table 3.2: Per data set summary information.

3.7 Data sets

The computation of the demands draws on several different data sets, as summarised in Tables 3.1 and 3.2. This section describes our approach for harvesting and preparing these various large data sets, each collected at a different location at a different granularity.

3.7.1 CDN Data

Using logs that feed into the CDN billing system of a major CDN provider, in our study Akamai [Aka], we extract the information which clients are accessing which Web object at which time. From this, we can deduce for each client set how much content from which publisher is accessed (after appropriate anonymisation). Each individual log file records all accesses to some part of the CDN infrastructure during some time period and is available for processing some time after the last recorded access. We captured logs for three two-hour time periods: 9–11:00 h UTC on Mon Apr. 26th, 2004 (CDN1) and 8:30–10:30 h UTC on Wed Apr. 28th, 2004 (CDN2) and 17–19 h UTC on Wed May 5th, 2004 (CDN3) from more than 90 % / 85 % / 65 % of all the operational servers of the CDN. There are two reasons why we did not capture logs from all servers: Logs for certain time periods arrive in chunks, thereby imposing huge instantaneous bursts

that sometimes can overload our limited research collection infrastructure. Other logs can be delayed due to remote network outages, and thus arrive after we stopped our data collection process. [4] In addition, the online collection is augmented by an offline retrieval of some subset of the logs via an archival system. We initially aggregated this data using the methodology described in algorithm 1 on page 44, using a time aggregation of half an hour. This time aggregation was chosen to examine the spatial, rather than the temporal variability of the data.

3.7.2 Data from three user sets

Three sets of client access information were extracted from packet level traces at the (then) 1 Gb/s upstream link of the Münchner Wissenschaftsnetz (MWN) in Germany. MWN
The MWN provides external Internet connectivity to three major universities (Ludwig-Maximilians-Universität München, Technische Universität München, Hochschule München) and a number of smaller universities, government organisations, and research institutes. Overall, the network contained about 50,000 individual hosts and 65,000 registered users at the time that the study was conducted. On a typical day, the MWN exchanges 1–2 TB of data with its upstream provider. On the 13th of May 2004 during the day (8–20 h), 295.5 GB used the HTTP port, which corresponds to 26.5 % of the traffic. During the night, 112.2 GB (18 %) of the traffic was HTTP. This indicates that the Web is still a major traffic contributor.

Our monitoring is realised via a monitoring port on a Gigabit Ethernet switch, just before the traffic passes the last router to the Internet. We captured the raw packet stream using `tcpdump` on disk, and then extracted the HTTP connections offline using the HTTP analyser of the intrusion detection system bro [Pax99]. We chose this approach over feeding the packets "live" into bro, since this way we can allow bro to consume more CPU time than real time. The resulting trace contains all relevant HTTP header information, and it is much more compact than the raw packet data.

Since extracting HTTP data at Gigabit speed is almost impossible using standard PC hardware [Fel00a], we split our client base into three groups: one for each university (TUM, LMU), and one that covers the other organisations (MISC). To ensure a reasonable coverage of all client groups, we monitored each client group for a 2-hour period, rotating through the groups. Accordingly, each trace captures all downloads of all clients in the group from all publishers as well as the CDN. In total, we collected 1,017 traces, each of which covers a 2-hour period. This approach ensures reasonable packet loss rates: Of the 1,017 measurement intervals, the number of intervals with more than 0.1 % / 1 % / 10 % packet drops (as reported by `tcpdump`) was 124 / 22 / 1. The maximum packet loss rate was 10.18 %, the average was 0.23 %, and the median was 0.0028 %.

[4] The relatively bad coverage for the May dataset is due to having to use a compute server for retrieving and storing the logs.

3.7.3 DNS data

We identified the IP addresses of roughly 7,000 DNS servers using a different packet level trace. The way that we identified these DNS servers ensures that each server supports recursive queries. The process, however, does not pay attention to the distribution of the DNS servers within the Internet infrastructure. Therefore, in the next step, we identified a subset of 516 DNS servers that return different results when resolving the name of the main CDN Web server. The 516 DNS servers are located in 437 ASes in over 60 countries. We restrict ourselves to using this subset, in order to reduce the load on the overall DNS system while achieving a good coverage of the Internet infrastructure. To resolve which publishers are using a distributed infrastructure, we selected a subset of 12,901 hostnames used by the publishers. The resolution of these hostnames resulted in more than 5.4 million queries of which 98.2 % received a valid response.

3.7.4 BGP data

We constructed a joined BGP routing table from the individual BGP tables on April 28th, 2004 from RouteView [Rou] and RIPE's RIS project [RIP]. Our joined routing table contains 161,991 routable entries. Furthermore, we extracted an approximation of the contractual relationships between the ASes using a methodology similar to that proposed by Gao [Gao00].

3.8 Experimental results

In this section, we present our results of applying our methodology to the various data sets discussed in Section 3.7.

3.8.1 Estimating CDN publisher demands

Our first step is to estimate how much traffic is sent by the CDN on behalf of each publisher to each client set. For the initial analysis in this chapter, we decided to use static groups of /24 prefixes to define client sets. We observe 1,130,353 different client sets within the datasets CDN1 and CDN2. This corresponds to a 23.6 % coverage of the overall IPv4 address space and a 52 % coverage of prefixes within the routable IPv4 address space. 1.3 % of the observed client space is not publicly routable, perhaps due to placement of CDN servers within private networks. In total, the client sets accessed roughly 41 TBytes of data via the CDN network. Thus, on average, each client set accessed about 36 MBytes over the three trace periods.

The Internet has obviously many client sets and a sizable number of publishers. But who is contributing the majority of the traffic—is it a small set of client sets, or a small subset of the publishers? Even by just studying the amount of traffic serviced by the CDN, we can get a first impression of these relationships. In Figure 3.5 on the next page, we rank client sets by total traffic received from the CDN from largest to smallest, and plot the percentage of the total traffic attributable to each for each 30 minute

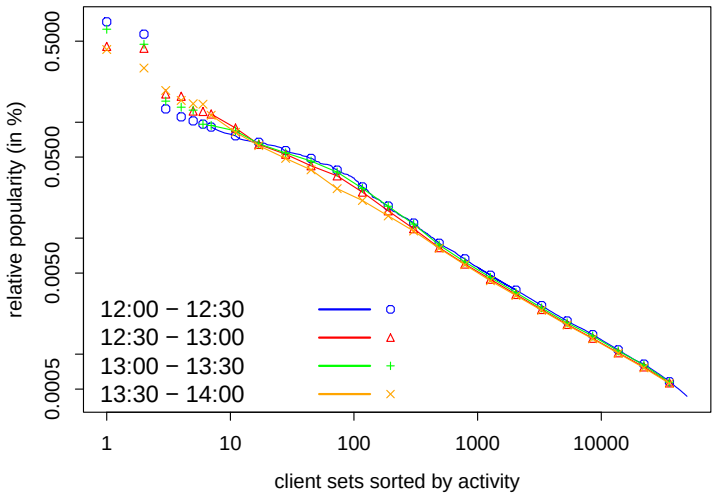


Figure 3.5: CCDF of client set traffic volume (% bytes served from all publishers each 30 min).

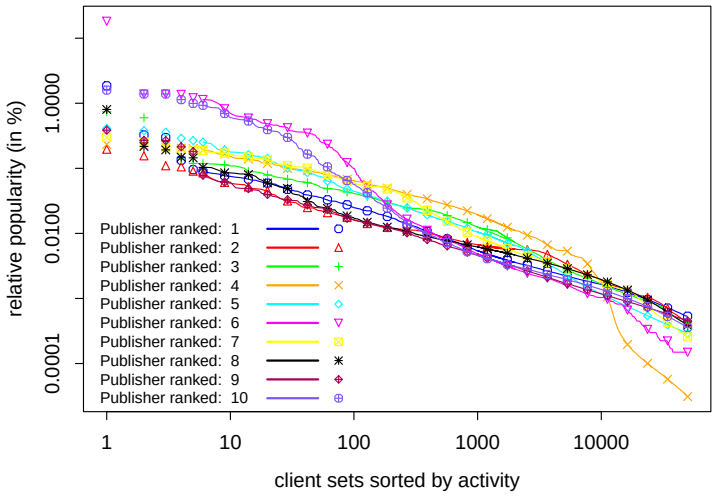


Figure 3.6: CCDF of client set traffic volume (% bytes served) per top-10 publisher during the two hour period of CDN2.

time interval of the CDN2 trace. This corresponds to plotting the (empirical) complementary cumulative distribution function (CCDF) of the traffic volume per client set. In order to not obscure the details in the curves we use lines instead of marking each point for ranks greater than five. To better distinguish the curves, we add some supporting markers. As predicted, we find a "linear" relationship on the log-log scale, an indication that the distribution is consistent with the characteristics of a Zipf-like distribution [Zip, BCF+99] (cf. Glossary). The client sets are sorted by their activity in terms of downloaded bytes; the first client set is the most active one. This implies that one has to look for the linear relationship in the left-hand part of the plot, while artefacts can be expected at the right-hand side.

But do client sets exhibit the same sort of activity distribution, even if we focus on *individual* publishers rather than on all publishers taken together? In Figure 3.6 on the previous page, we explore the characteristics of the top 10 publishers, selected by the total number of bytes that they serve to all client sets (using the same plotting technique as before). The fact that we still observe a "linear" relationship on the log-log scale indicates that even single publisher demands are dominated by the behaviour of a few client sets. One aspect that may be contributing to these effects is that client sets are located in different time zones: About 40.4 % of the client sets in CDN1 and CDN2 are located in the US, 9.4 % in Japan, 6.0 % in Korea, 4.2 % in the UK, 4.2 % in China, 3.9 % in Germany.[5] One reason for reduced demands is that in some client groups, most users are sleeping, while users of other client sets are at work, etc. Although the impact of time zones has to be explored further, we start by subselecting various subsets of client sets. Each of these client sets covers either one (Japan), two (UK, France, Germany), or four time zones (US). We still observe activity drops that are consistent with Zipf-like distributions (plots not shown) if we split the demands per client or per time. The bends for Publishers 6 and 10 in Figure 3.6 are due to the superpositions of accesses by client sets in the US and abroad; the ones in the US have a higher demand than those outside the US.

Even though the client sets in Figure 3.6 are ranked separately, according to their activity for each publisher, it also shows that a client set that receives the most bytes from one publisher does not do so from another publisher. Rather, there are significant differences. This indicates that each publisher in the Internet has to determine for itself who the heavy hitters (contributors) among the clients are—extrapolating from one client set to another can, in fact, be misleading.

But what is the behaviour if we consider the data from the viewpoint of the client sets? In Figure 3.7, we explore the popularity of content served by the CDN on behalf the publishers (using the same plotting technique as before). Again, we observe a curve that indicates a Zipf-like distribution in the range of 1–1,000. The drop-off in the curve for less popular publishers indicates that there is a large number of publishers that do not serve a lot of data via the CDN. However, this does not disprove that, for the popular publishers, the distribution is consistent with a Zipf-like distribution.

[5] The mapping of network to country was done using an internal Akamai tool called `EdgeScape`.

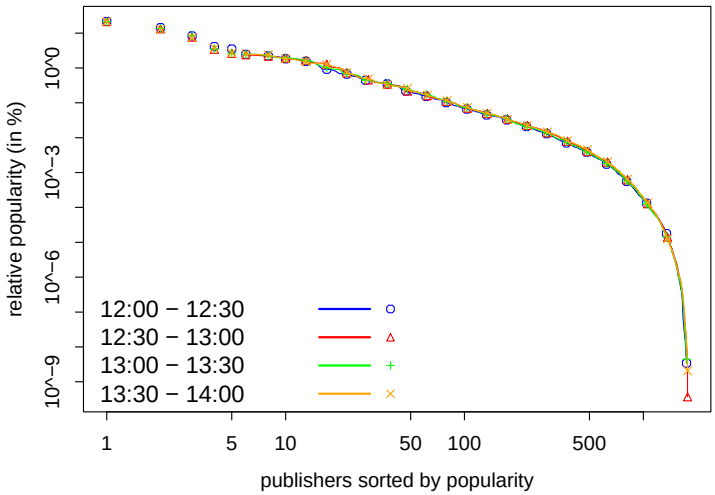


Figure 3.7: CCDF of publisher traffic volume (% bytes served to all client sets each 30 min).

Generally, we observe the same kind of curves for all data sets and for each subset of the datasets. For example, in Figure 3.7, the curves for the publisher popularity in terms of traffic volume between consecutive 30-minute time periods fall on top of each other. The same observations hold if we look at the individual publishers or the client sets over consecutive 30-minute intervals. But this does not imply that it is always the same publisher or the same client set that dominates the distribution. Accordingly Figure 3.8 plots the bytes contributed by each country during one 30-minute time period vs. another 30-minute time period. The left plot does so for consecutive time periods. The nice concentration around the diagonal indicates that the volume changes are not rapid within any of the three datasets. In contrast, the right plot shows the same kind of plot comparing corresponding 30-minute time periods from the 26th of April to those of the 5th of May. (A 30-minute time period starting at offset x in one trace corresponds to the 30-minute time period starting at offset x within the other trace.) Note that, due to the time shift, one should expect a larger spread. This is indeed the case, indicating that the popularity changes have to be considered not being just time-of-day variations.

3.8.2 Estimating relationships between CDN and publisher flows

Once we know how much Web traffic is flowing from the CDN to each client set, we need the ratios from the packet level traces to extrapolate from the partial CDN publisher demands to the Web publisher demands. Accordingly we apply our methodology to the client access logs. Note that we are not necessarily capturing all of the traffic from the publisher, since our methodology is based on the referrer fields in the requests

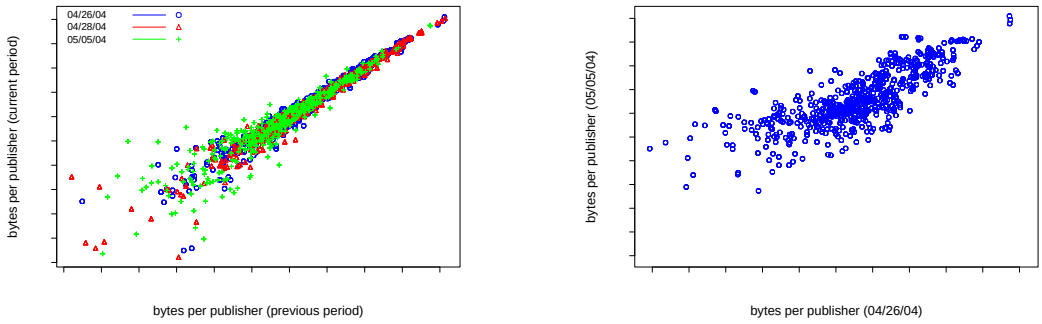


Figure 3.8: Scatterplot: publisher bytes for time period t vs. period t'.

Users	Description	Requests (in K)		Bytes (in GBytes)	
		absolute	relative	absolute	relative
TUM	Total	357,621	100.00 %	3795.83	100.00 %
LMU	Total	91,104	100.00 %	721.60	100.00 %
MISC	Total	62,013	100.00 %	636.47	100.00 %
All	Total	510,738	3 × 100 %	5153.90	3 × 100 %
TUM	CDN	15,065	4.21 %	119.00	3.14 %
LMU	CDN	4,449	4.88 %	26.75	3.71 %
MISC	CDN	3,043	4.91 %	27.40	4.31 %
TUM	CDN customer	10,650	2.98 %	83.95	2.21 %
LMU	CDN customer	2,549	2.87 %	13.75	1.91 %
MISC	CDN customer	2,107	3.40 %	11.20	1.76 %
TUM	related non-CDN	6,121	1.71 %	44.61	1.18 %
LMU	related non-CDN	1,325	1.45 %	5.15	0.71 %
MISC	related non-CDN	1,212	1.76 %	4.91	0.77 %

Table 3.3: Basic statistics of the user access characteristics.

for CDN-delivered objects—i. e., there might be even more CDN customer data being delivered than we are estimating.

We start with presenting some basic characteristics of the data sets from the three client populations covering all monitored subnets, see Table 3.3. Overall, in the TUM, LMU, and MISC data sets, we observed roughly 522 million different requests for Web objects for more than 5.15 TBytes of data. This implies that the mean object size in our data sets is about 9.5 kBytes. The mean size of an object served by the CDN to the clients at TUM, LMU, and MISC is a bit smaller at about 8 kBytes. This accounts for the difference between the percentage of requests directed towards the CDN vs. the percentage of bytes. Although 4.2–4.9 % of all HTTP requests are served by the CDN, this corresponds to only 3.14–4.31 % of the HTTP bytes.

From Table 3.3, we see that the clients only retrieve 1.8–2.2 % of the HTTP bytes from the CDN customers' own servers. This indicates that the ratio of bytes served by the CDN vs. the bytes served directly by the publishers can vary from 1.4 to 2.5: The relative percentage of requests directed to the CDN customers is larger than the relative

percentage of bytes retrieved from the CDN. This indicates that CDN customers delegate their larger content to the CDN, which is to be expected. Although publishers offload a large amount to the CDN, they normally do not delegate all of their traffic—rather, some of their traffic is served from servers that belong directly to the publisher, as we have seen in the packet level traces. Therefore, our approach for estimating publisher traffic can be expected to yield estimates of interesting interdomain traffic flows for a significant fraction of the overall traffic volume.

The fraction of bytes in the category *related to non-CDN-customers* gives us another possible avenue for estimating interdomain traffic flows. There are two reasons why requests (or traffic) falls into this category: publishers offload some of the content to other service providers (e. g., those providing targeted advertisement), and some of the publisher's content is served in connection with other sites (e. g., advertisements on someone else's Web page). We note that a large number of Web advertisement providers exist, most of these providing a decentralised infrastructure. Thus, even though the approach of analysing their traffic as well may indicate some additional potential, we considered the overhead in determining identities and services of these advertisement publishers was too large in relation to the potential benefit. We thus solely focused on the ratio of traffic served by the CDN on behalf of a publisher vs. the traffic to the publisher itself.

For this purpose we need to associate the bytes served by the CDN and the bytes served by CDN customers' own servers with the appropriate publisher. Using information sources that are internal to Akamai, we were able to identify 23 million requests from the MWN for Akamai-hosted URLs (Table 3.1). Although 23 million requests are a sizable number, the individual number of requests for objects served by the CDN over smaller time period (2 hrs) are significantly smaller. Averaged over the whole duration of the trace collection, this implies that one can expect to see only 2,000–20,000 requests in each data set for each two-hour time period. Just averaging is unfair, of course, since there will be many more requests during busy hours than during off-hours, e. g., in the middle of the night. In addition, some subnets (e. g., those with Web proxies) generated many more requests than others. Nevertheless it points out the problem of observing enough samples for deriving a reasonable ratio estimate.

Here we receive help from a fact that has been observed in many other contexts: some publishers have much more popular content than others. We rank the number of requests (Figure 3.9) and bytes (Figure 3.10) by provider from the largest to smallest for both data sets, and plot the percentage of total requests (or bytes, respectively) attributed to each. For those publishers that contribute a significant percentage of the bytes, these curves are "linear" on a log-log scale. Again, this characteristic is consistent with a Zipf-like distribution. Together, these two observations imply that we can expect to find time periods with a reasonable number of observations for some significant subset of the publishers in our user access data sets. We now focus on those $(\textit{time period}, \textit{publisher})$ pairs with enough observations.

Here we define "enough" as observing at least 50,000 requests satisfied by the CDN on behalf of a publisher and 500 requests served by each publisher itself per aggregation

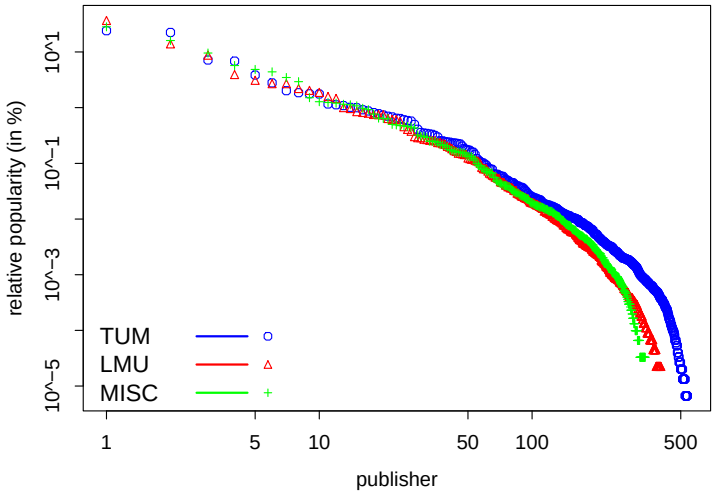


Figure 3.9: CCDF of requests per publisher.

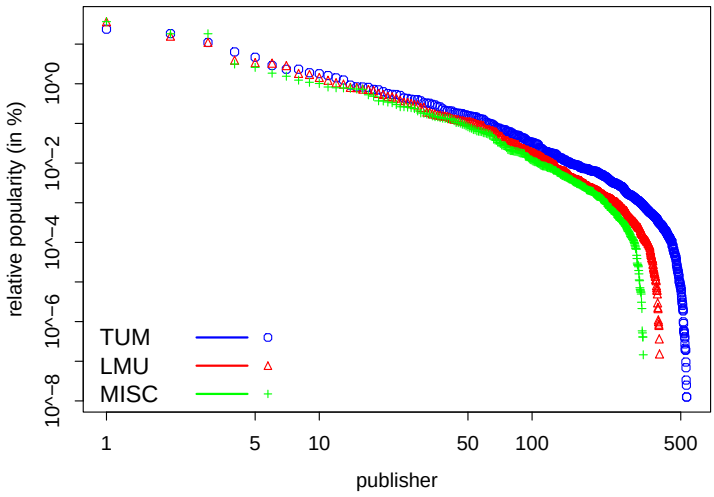


Figure 3.10: CCDF of bytes per publisher.

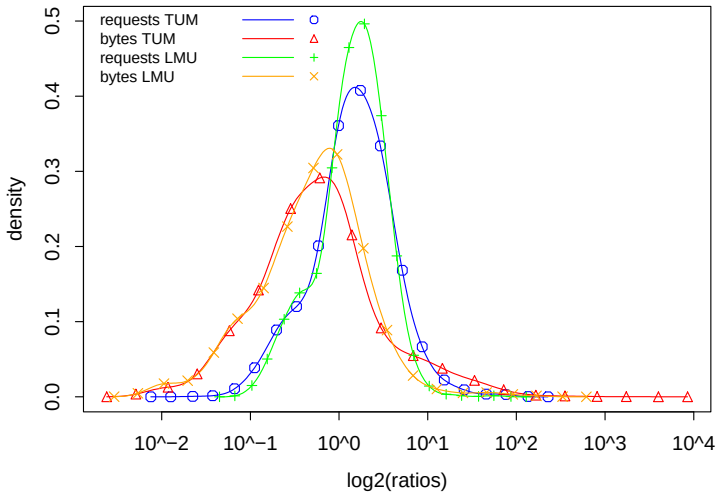


Figure 3.11: Density of log_2 of requests and bytes ratios for objects of Akamai customers requested from TUM and LMU. Each ratio is calculated as $\frac{\#\textit{ requests or bytes Akamai servers}}{\#\textit{ requests or bytes publisher-owned servers}}$.

period.[6] Using these selection criteria, we compute the ratios of bytes for each publisher and each aggregation period. Not too surprisingly, we find that the ratios span quite a wide range of values: from 0.01 to 100.

Comparing ratios is awkward: Is, e. g., the "difference" between 0.03 and 0.06 the same as the "difference" between 16 and 32? In this context, the answer is yes, since both "differ" by a factor of 2. Therefore, to ease comparisons of ratios we, in all further discussion, will use the binary logarithm of the ratios. Accordingly, 0.03 is transformed to -5, and 0.06 to -4, and 16 to 4, and 32 to 5. Now the differences in both cases are 1.

Figure 3.11 plots the density of the transformed ratios for the TUM and LMU data sets for both bytes as well as requests. We observe for all data sets that the ratios span a significant range of values from -10 to 10, both for requests as well as for bytes. This indicates that different providers use different policies with regards to delegating their information to the CDN. We see furthermore, as expected, that the CDN usually provides more bytes than the original publisher for most, but not all publishers. In addition, with regards to requests, the distribution is more balanced. This indicates that some publishers use the CDN for big objects, such as software distribution.

Even though the overall distribution of the ratios is interesting, more relevant for the purpose of estimating the publisher demands is the question: How stable are the ratios across time and user populations? Overall, it is well known that traffic volume [WPT98] and flow arrival streams [Fel00b] are self-similar and exhibit significant

[6] Using a value of 500 is fairly arbitrary.

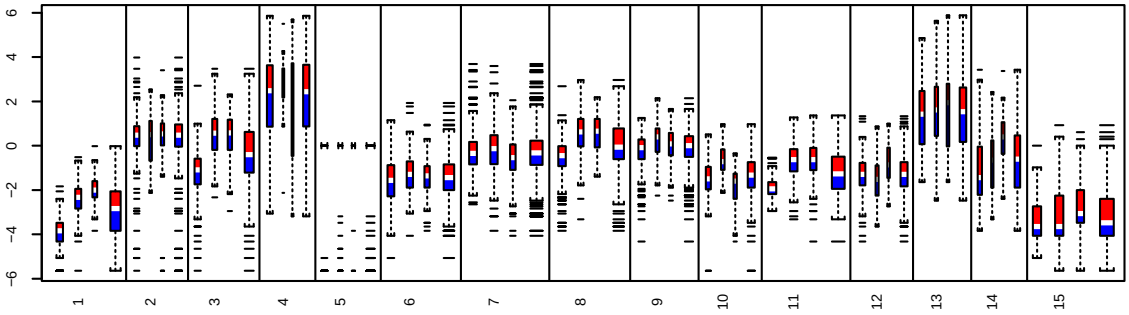


Figure 3.12: Box plot of $log_2(\text{ratios})$ for the 15 most popular publishers (labelled 1, 2, ..., 15). For each publisher results for four data sets are shown (from left to right): TUM / LMU / MISC / (TUM ∪ LMU ∪ MISC).

burstiness. Therefore, we can expect some fluctuations with regards to the number of requests over time. In addition, not every user will access the same pages from the publisher, and different subsets of pages will lead to different ratios in terms of bytes from the publisher and the CDN. But what are the impacts of all these causes of instability? Our estimation methodology allows us to explore the size of these instabilities since it will yield multiple samples of estimated ratio values for various publishers. Figure 3.12 shows box plots of the ratios for the 15 most popular publishers for the samples of the three data sets, TUM, LMU, and MISC. Box plots can be used to display the location, the spread and the skewness of several data sets in one plot: the box shows the limits of the middle half of the data; the line inside the box represents the median; the box widths are proportional to the square root of the number of samples for the box; whiskers are drawn to the nearest value not beyond a standard span from the quantiles; and points beyond (i. e., outliers) are drawn individually.

Most of the boxes have a reasonably small spread (less than two), whereas others have a sizeable spread, e. g., those at index 4. This is partially due to a fairly small sample size and partially due to the variability of different content that is offered by that publisher. Further aggregation and combining the information from different user sets can sometimes be helpful—Figure 3.12 also shows the box plots for the samples from the combined data sets. Although some estimations of the ratios stabilise, as indicated by the smaller range of the box, others expand due to the differences in the user behaviour.

Generally, we can estimate the ratio of publisher demand serviced by the CDN vs. that serviced by the publisher. But there are drawbacks to this approach: A large number of requests needs to be monitored in order to derive reliable estimations. The estimations can vary across time and some attention has to be paid towards different

subject/interest areas by different user sets. Furthermore, not all user sets will access sufficiently many objects from all publishers that are customers of the CDN. Therefore, this approach should be combined with other approaches for estimating the ratios, e. g., static exploration of the Web site and information from the publisher itself.

3.8.3 Mapping of publisher demands to Web traffic demands

The next step is to apply our methodology for mapping the publisher demands to Web traffic demands. The open question is: how well does the proposed methodology of mapping each client set and each hostname to a single server IP address work? We propose a two-step process to address this issue. First, we need to identify the set of IP addresses for each hostname. Then we need to identify which subset of the IP addresses to choose for each client set, since it may be the case that not all client sets use all server IP addresses.

If a hostname is hosted by the CDN itself, or if the infrastructure is using DNS round robin by itself, the identification step is simple. In the first case, we know which IP address serves the traffic; and in the second case, all returned IP addresses are used. Using the data described in Section 3.7.3, we observe that of the 12,901 hostnames, 2,106 (16.3 %) are hosted by the CDN itself, while 1,242 (9.6 %) are using some form of proximity-aware load balancing, and 10,906 (84.5 %) are consistently returning the same set of IP addresses. Of these hostnames, 9,124 (83.8 %) are returning a single IP address whereas 1,079 (8.4 %) are utilising only DNS round robin. Most of these (830) are using two IP addresses, while 79 are using more than five IP addresses. Therefore, we have solved the problem for 90.4 % of the considered hostnames. If most publishers are serving their content out of a small number of servers, then most clients must be far away from those servers, which indicates that a significant fraction of the traffic that we capture will be interdomain traffic.

This leaves us the with 1,239 hostnames that are hosted on a distributed infrastructure and for which proximity-aware load balancing is used. To better understand this infrastructure, we show a histogram of the number of IP addresses (Figure 3.13) and the number of ASes (Figure 3.13). We observe that most of these hostnames (83.5 %) are only mapped to a small number of IP addresses ($\leq$ 5). Indeed, more than 34.7 % are using only two distinct IP addresses. Next, we examine whether the infrastructure crosses domains (see Figure 3.14). 377 (30.4 %) of all hostnames that use proximity routing are in a single AS. This means that, from the view point of interdomain routing, we will not be able to distinguish these demands. We observe that 44 % of the hostnames are located in at least two, but at most five different ASes.

To explore how the infrastructure of the remaining 862 hostnames is embedded in the Internet, we studied the minimal AS distances of the ASes of the IP addresses of the distributed infrastructure to the ASes of 500 randomly selected IP client sets. In order to compute the distances, we consider the contractual relationships as derived from the routing tables [Gao00]. Each AS path may only cross a single peering/sibling edge, and may never follow a customer-to-provider edge once it has followed a provider-to-peer edge. Any edge unclassified by the heuristic is treated as a “sibling/peer” link.

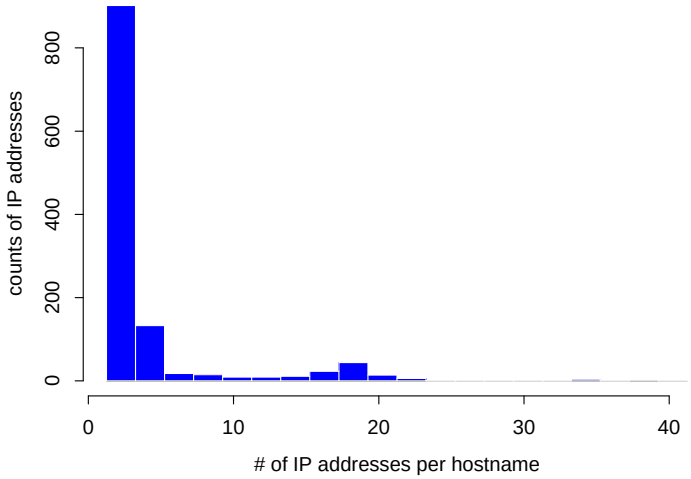


Figure 3.13: Distributed infrastructures: IP addresses per hostname.

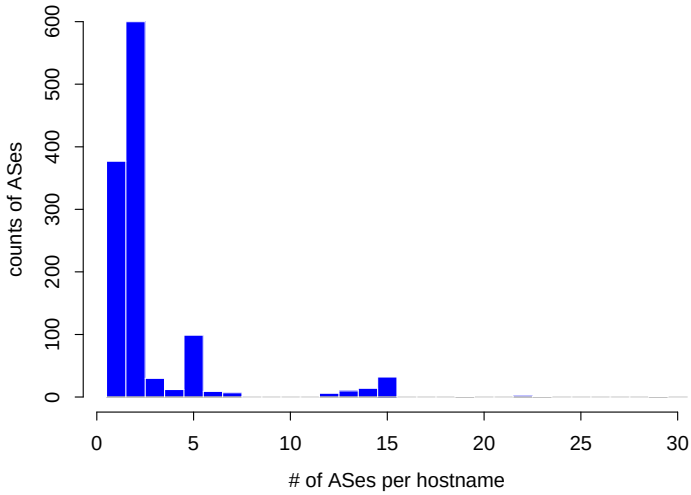


Figure 3.14: Distributed infrastructures: ASes per hostname.

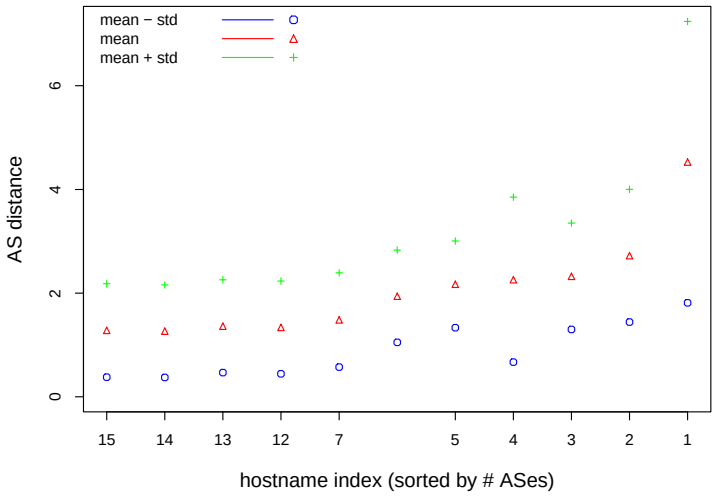


Figure 3.15: Distributed infrastructures: AS distance between client sets and publisher hostnames.

We observe (Figure 3.15) that providers that use more servers and distribute them in various AS indeed gain some benefits. The mean distance and the standard deviation to other ASes is reduced.

3.9 Summary and open questions

In this chapter, we have proposed two models for interdomain traffic demands, publisher demands and Web traffic demands, that capture the origin, the volume, and the destination of the data, and thus provide an interdomain traffic matrix for Web traffic. We believe that this simple abstraction can facilitate a wide range of engineering applications, such as traffic engineering, planning of content delivery, or network simulation. We have further presented a methodology for populating parts of the demand model using logs from CDN networks, observations from user sets, the DNS, and the routing system.

The experimental results obtained by applying our methodology to logs from a major CDN and user traces from two large user sets are promising. Our approach seems to allow us to capture a significant fraction of all Web traffic. Viewed on any scale, but particularly in terms of the number of pairs, our matrices are some of the largest ever generated. We have demonstrated that it is indeed possible to combine server log data from a CDN with packet level traces from large user sets to estimate a good chunk of all interdomain Web traffic as proven by the diversity and coverage of the demands. Nevertheless, our results (especially the numerical estimates) should be treated as pre-

liminary and viewed mainly as an indication of the potential of the methodology. We present a collection of directions for further research:

1. We have captured only one class of traffic, namely HTTP. Even though several studies have shown that HTTP traffic is among the most common, its dominance has recently been challenged by new classes of traffic such as peer-to-peer file sharing data and streaming media. How well does HTTP traffic demand effectively represent overall traffic demand? How can traffic demand for other classes be estimated?

2. In this work we assume that the number of bytes served by the content provider for each Akamai-served object can be estimated by examining traces from a small number of large client sets. Is the observed ratio of bytes served by the customer to bytes served by the CDN (reasonably) invariant across diverse user sets? At this point we have examined only two. It is possible that content providers tailor their Web pages for different client sets individually; e. g., a U.S.-based site might choose to serve more compact (fewer bytes) Web pages to overseas clients.

3. Now that we have a means of estimating interdomain traffic demands, we are beginning to explore aspects such as temporal (time-of-day) and spatial distributions and analyses of publisher/user dynamics. But we expect it to be even more fruitful to combine this data with routing information, specifically BGP tables. How does BGP respond to network bottlenecks? How do the demands shift in response to routing changes?

4 Characteristics of Web search related traffic

After we have analysed Web traffic in general on a global scale in the previous chapter, we now focus on the characteristics of Web traffic that is the direct or indirect result of search engine usage. Search engines are a vital part of the Web and thus the Internet infrastructure. Interactions with search engines make up a tremendous part of users' Web activities. Therefore understanding the behaviour of users searching the Web gives insights into trends, and enables enhancements of future search capabilities. Moreover, the characteristics of Web-search induced traffic can be of use in other domains, e. g., interdomain traffic matrix estimation [CJMW05]. Indeed, search engines are an active research area in itself.

Similar to our methodology for estimating Web traffic demands, one possible data sources for studying Web search behaviour are either server side logs or client side logs. Unfortunately, current server side logs are hard to obtain, as they are considered proprietary by the search engine operators. Therefore we in this chapter present a methodology for extracting client-side logs from the traffic exchanged between a large user group and the Internet. The added benefit of our methodology is that we do not only extract the search terms, the query sequences, and search results of each individual user but also the full *clickstream*, i. e., the result pages users view and the subsequently visited hyperlinked pages. We propose a finite-state Markov model that captures the user Web searching and browsing behaviour and allows us to deduce users' prevalent search patterns. To our knowledge, this is the first such detailed client-side analysis of clickstreams.

4.1 Introduction

Besides gaining new insights into user search patterns, query clickstreams can serve as a means for Web search enhancement. In the past, query logs [LW04] have been used to extend state-of-the-art link analysis on the Web graph, or to perform query clustering [CWNM03] for query expansion. The implicit feedback inferred from such logs can be used as input to machine learning approaches [RJ05], or in the estimation process of language-model based information retrieval [STZ05].

Unfortunately, the currently available data sets about clickstreams are rather limited. In principle, there are two ways of gathering such data: either on the server, or on the client side. As server side data is considered proprietary, current analyses are limited to only a few search-engine-specific data sets, including one gathered in 1998 from Altavista [SHMM98], one from the Excite search engine [SWJS01] in 1997, and one from

Vivisimo [SKP+05] in 2004. Furthermore, none of these data sets include the full clickstream which consists of all user accesses to Web pages related to a search query. The client-side data gathering has focused on asking volunteers to surf the net using additional browser plugins, e. g., [WOHM06], or enhancing HTTP proxies with extended logging functionality, e. g., [AWS06]. Yet, not all sites currently use proxies or are willing to modify them.

Our approach is to extract client-side logs from packet level traces of the traffic exchanged between a large research network and the Internet. From the packet level traces, we extract all HTTP requests and responses, as well as the bodies of all responses from search engines. From this data we reconstruct each of the users' query sessions. This includes that we determine for each search query the position of the search results that the user clicked upon (if any). Furthermore, we recursively analyse how many links (if any) the user followed from the search result.

Utilising data obtained at the border of the MWN, the large scientific network located in Munich which we mentioned in Section 3.7.2 on page 49 already, we present a characterisation of the query sessions as well as a finite-state Markov model that relates the Web search clickstreams to the Web hyperlink structure.

4.2 Related Studies

Studies of Web search behaviour can be categorised along different axes according to whether the data is gathered from search engine logs or on the client side, the time period they cover, as well as the measures applied and research questions pursued. Jansen and Pooch [JP01] survey and compare studies on traditional information retrieval systems, online public access catalogues, and Web searching up to the year 2000. They find that there is a lack of clarity in the descriptions and that the use of different terminologies by the various studies make the results hard to compare. They propose a framework for such analysis, which we use in this study.

More recently, a number of researchers, e. g., [LLC05] have focused on categorising queries according to user search goals in order to improve search performance. Lee et al. [LLC05] rely on packet level traces. Yet, as the focus of their paper is on automatic identification of user goals in Web search, they do not systematically establish a relationship between the position of the search results and the gathered clickstream, nor do they consider follow-up clicks. Another line of work (e. g., [BJC+04]) aims at a topical query classification using data from a major commercial search service.

Chau et al. [CFS05] examine which documents of the result pages are viewed by the user; they, however, do not consider which hyperlinks the user follows beyond these. In addition, their study is limited to Web site search. Spink et al. [SJO00] use a Markov model for query reformulation patterns of the Excite search engine. Their model, however, cannot include the user behaviour beyond the search engine interface: It neglects which documents that are listed on the result pages are visited by the user during a query session. Thus, none of these studies take the whole Web query clickstream into account.

4.3 Terminology

To simplify the discussion, we briefly summarise some of the terms that we use in the remainder of this chapter. The definitions are in part taken from Jansen et al. [JP01] and from Spink et al. [SWJS01].

Term: any unbroken string of alphanumeric characters entered by a user, e. g., words, abbreviations, numbers, URLs, and logical operators. — term

Query: a set of one or more search terms as typed by the user (may include advanced search operators). After a query has been issued, the search engine (hopefully) returns a *query result page*. — query; query result page

Query session: a time-contiguous sequence of queries issued by the same user.[1] — query session

Unique query: a query that is unique within one query session. — unique query

Repeat query: re-occurrence of the same query in the same session, e. g., if a user retrieves several result pages for one query. We furthermore distinguish between *next result page queries* and *real repeat queries*: The latter indicate multiple requests for the same query result page. — repeat query; next result page query; real repeat query

Result links: links contained in the query result page. — result links

Result position: the absolute position of a result link in the query result page. — result position

Clicks: `text/html` HTTP requests that are the result of a click on a hyperlink. — click

Result clicks: result links on which a user clicks. — result click

Clickstream: all `text/html` requests related to a query session. — clickstream

4.4 Search clickstreams

In order to monitor the *Web search clickstream* of a set of users, we rely on capturing client side logs. More specifically, we suggest to use packet level traces as our main data source. From these traces we extract:

for the search engine under study: all HTTP requests and responses including their bodies and the HTML links they contain.

for all Web servers: all HTTP request and response headers.

[1] An alternative definition for a query session is "a sequence of queries by one user which satisfy a single information need." Unfortunately, identifying such sessions is only possible by finding semantic demarcations, e. g., by relying on query similarity. But recent work [SKP+05] indicates that such demarcations may not exist, as a user may work simultaneously on several information needs, or may have rapidly switching information needs, or his information needs may evolve or shift during his search activities.

Neither standard browsers nor Web proxies provide us with this kind of data. Thus one would have to either instrument all Web browsers or install a modified Web proxy into the data path [AWS06, KR01].

Although one could use any tool that can reconstruct HTTP level detail from packet level traces (see, e. g., [KR01]), we utilise the HTTP analyser of bro [Pax99], a network intrusion detection system. A self-written "policy" file for bro extracts all the data described above from both standard HTTP/1.0 as well as persistent or pipelined HTTP connections. For extracting HTML links from the bodies retrieved from the Web search engine, we use the Perl module `HTML::Parser` version 3.45 from [Net].

4.4.1 Search queries

To determine which requests are search queries, one has to consider the specifics of the search engine. We, in this chapter, focus on queries to Google, but the same principle methodology can be used to extract clickstreams for other search engines. We consider an HTTP request to be a Google query if the following conditions hold:

- The request is made to one of our locally-seen Google servers in the subnets `66.249.64.0/18`, `64.233.160.0/19`, `216.239.32.0/19`, `72.14.192.0/18`, or `66.102.0.0/20`.
- The `Host` header contains the string "`.google.`".
- The URI starts with `/search?`, contains a non-empty CGI parameter `q=...`, and does *not* contain the string `client-navclient-auto`, since we conjecture that these come from requests issued by a software, but not from direct user interaction with the browser.
- The `Content-Type` field of the response contains the string `text/html`.
- The `User-Agent` field does not indicate an automated query, i. e., it does not contain any of these strings: *bot, agent, crawler, wget, lwp, soap, perl, python, spider* (case-insensitive match).

4.4.2 Sessions (User Web search clickstreams)

In order to statistically evaluate the user search behaviour, we somehow need to group the search-related actions of individual users. Thus we identify search requests, and we determine individual *sessions* that contain all search requests and all other `text/html` requests that are directly or indirectly reachable via the result pages of the query requests (*search-induced clicks*).

The grouping of requests into sessions is shown in Fig. 4.1 and works as follows: For each arriving request, we determine whether it is a search request, using the criteria from Section 4.4.1. If this is the case, we utilise the fact that Google embeds a cookie called `PREF` in the search request stream. This cookie contains a portion labelled `ID`, which Google apparently uses for tracking individual users. We search in our pool of

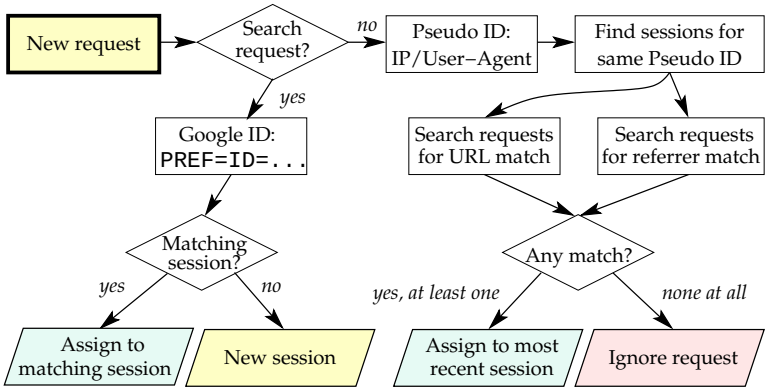


Figure 4.1: Determining the session for a new request.

currently active sessions for a session that matches the given `ID`, and add the request to the session. If no session matches, we instantiate a new session.

If the current request is not a search request, we determine client IP and HTTP `User-Agent` as a *pseudo-ID* (note that this concept is orthogonal to the Google cookie `ID`s). In all sessions matching this *pseudo-ID*, we look for requests that have either the same URL as the current request, or whose URL matches the HTTP referrer of the current request. We assign the current request to the session with the most recent matching request. If no match can be found, we ignore the current request.

Sessions that have not seen a request for more than five minutes are considered to have ended. They are removed from the pool of active session and will not be reactivated.

4.4.3 Query terms

Given that we can now group queries into sessions, we also want to examine how the queries within each session differ. Therefore we consider the *query terms* within the query based on an understanding of the specifics of the query language provided by the search engine. Google's basic search [Ggl] requires *all* search terms, except stop words[2], to be present in the results, i. e., all terms are implicitly combined by "and". Capitalisation plays no role, as all typed letters are automatically converted to lower case. In contrast, term order is decisive. Accordingly, we normalise the queries to lower case but retain the search term order. Negative terms, i. e., terms that should not occur, are preceded by a minus. Phrases are surrounded by quotation marks and are treated as a single term. A plus sign in front of a term tells Google that this is not a stop word; a stop word would be removed otherwise. The tilde sign before a term tells Google to include synonyms and is therefore kept. The "`site:`" operator restricts the search space to the specified domain. Even though Google offers several additional advanced features we, in this study, restrict ourselves to the ones discussed above, the most common ones.

[2]Stop words are very frequently occurring words that carry little information like *a, the, is, ...*

4.5 Search characteristics

We now describe the dataset that we use for our analyses of user Web search behaviour. The description is followed by an analysis of the user query behaviour, in order to reveal some characteristics of the user population that we examine.

4.5.1 Dataset

For our analysis of the clickstreams we use data collected from the *Münchner Wissenschaftsnetz* (*MWN*; Munich Scientific Network). At the time that we conducted our study, the MWN provided a 10 Gbit/s singly-homed Internet connection to roughly 50,000 hosts at two major universities along with additional institutes; link utilisation is typically only around 200–500 Mbit/s in each direction. Our monitoring machine is connected to the monitoring port of the border switch. Since MWN as a whole imposed too much load on our tracer machine running the bro HTTP analyser, we restrict our data gathering to Ludwig-Maximilians-Universität, the larger one of the two universities in Munich (about 44,000 students).

bro's memory consumption increases slowly over time, as it accumulates state from, e. g., dangling TCP connections, etc. Thus we start a new bro process every 45 min and let it run for 50 min. The resulting 5 min overlap ensures that HTTP requests that stretch over the 45-minute boundary are not lost, apart from a few long-lasting persistent connections. Duplicate records are removed.

We consider all requests issued from Thursday, August 17th, 2006, 17:07 MET until Friday, September 1st, 2006, 21:00 MET, excluding a monitor downtime period of about 18 hours. The median packet loss as reported by bro and LIBPCAP during each 50 min interval is 0.0 % (average: 0.15 %; maximum: 7 %). The total number of HTTP requests on TCP port 80 is 125,104,884; the number of transferred HTML objects is 28,026,595, of which only 19,601,616 have HTTP status code 200. Note that these numbers also include "abuses" of the HTTP protocol by non-Web applications.

During this time period, we identified a total of 545,455 Google search queries. There are 275 empty queries where the user clicked the "Search" button before entering a query. Out of the remaining Google queries, 414,184 are unique queries, and 130,996 are repeat queries. The repeat queries consist of 105,683 real repeat queries and 25,313 next result page queries. Manual inspection shows that most queries relate to local specifics of the area of Munich (see also Table 4.5.2 on page 70). Yet, as expected, the queries also reflect the academic environment. We note that, even though manual inspection reveals that many obviously recreational queries are made, our data is strongly biased towards academic users, and thus may expose a behaviour that is different from the general population.

4.5.2 Query session characteristics

To understand the characteristics of the sessions within the reconstructed clickstream, we start this section by presenting statistics similar to those of previous studies [CFS05, JP01, SWJS01].

Query length

The maximum number of terms per query we encountered is 199; one user copied a whole text snippet into the query box. The median query length is 1 while the mean is 1.67. This shows an even stronger trend towards very short queries than observed in the course of previous studies, e. g., [SHMM98]. 64 % of the queries consist of a single query term, 83 % contain less than three terms, and 99 % consist of less than six terms.

Use of search operators

Of the 414,184 unique queries, 11,977, i. e., 3% use phrase search. 832 queries enforce the occurrence of a search term via the plus operator, and 315 queries use the minus operator to exclude search terms. However, similarity search utilising the tilde operator does not occur at all. Finally, domain queries (restricting the search to a specific Web site) occur 492 times. To summarise, only about 0.4 % of the queries made use of features other than phrase search.

Terms

We identify 249,445 distinct terms in our data; however, 124,095 of these are of the form "info:<*url*>" to request further information on specific URLs from Google. We omit these terms in the following statistics (note that this might be an explanation for the large number of one-keyword queries found). The most frequent terms are listed in Table 4.5.2 in descending order of their frequency. This statistic clearly reveals a bias in our dataset towards local themes (high frequency of terms relating to Munich and Germany), as well as academic subjects. For example, the term “lmu” is short for “Ludwig-Maximilians-Universität”, which is the university whose Web traffic is analysed in this study. Also note the use of stop words (“in”, “der”) and the set of presumably navigational queries, e. g., “wikipedia”. In addition, queries relating to recreational activities are quite frequent such as “hotel”, “wetter” (which is German for “weather”), as well as searches related to companies and products (e. g., both “siemens” and “xp” occur with frequency 224). Yet, in spite of the academic environment, we also find queries on pornographic material, such as “sex” (387) and “porn” (116). Interestingly, the prominence of the term “+” possibly indicates that users have trouble with the correct usage of the “+” operator, and write “+␣the” instead of “+the” to prevent “the” from being neglected as a stopword, or that they use it for phrases such as “tom + jerry”.

Query sessions

We identify 153,719 query sessions. The median number of queries per session is 2, the mean number is 3.5. There are sessions with more than 100 queries, but more than 46 % / 20 % only contain one / two queries. Similarly, the median number of *unique* queries is 1 and the mean number decreases to 2.7. Accordingly, the median number of repeat and real repeat queries is 0 while the mean is 0.85 and 0.69, respectively.

Term	Freq.	Term	Freq.	Term	Freq.
münchen	9,815	lmu	1,009	die	710
in	2,830	für	969	windows	703
the	1,875	2006	895	wetter	676
of	1,809	online	852	a	662
+	1,724	von	832	lyrics	628
der	1,532	de	825	java	623
und	1,405	to	808	uni	599
download	1,373	bayern	799	berlin	595
muenchen	1,370	linux	783	free	591
and	1,020	wikipedia	761	hotel	582

Table 4.1: The 30 most frequent search terms.

Query refinements

To describe the relationship between two consecutive queries in the same session, we distinguish between the following kinds of query modifications: *repeat* (the same query again), *disjoint* (no overlap in the query terms), *add* (the follow-up query is a superset of the previous one), *delete* (the follow-up query is a subset of the previous query), and *replace* (the follow-up query and the previous one overlap but are not strict subsets). We find that there are 20 % (76,541) repeat, 68 % (265,715) disjoint, 4.75 % (19,661) add, 2.5 % (9,521) delete, and 4.75 % (20,023) replace modifications. This rather coarse-grained categorisation leaves space for further investigation, e. g., to examine how often terms are changed into phrases, or how often users re-order terms.

Result pages viewed:

The maximum number of viewed result pages is 32. The mean number is 1.06 and the median is 1. If we consider the unique queries, 96.5 % look at only a single result page. Less than 0.1 % consider four or more distinct result pages.

4.6 A state model for Web search

To better understand the behaviour of users that search the Web, we model Web user behaviour during a search session as a Markov model. The Markov model relates the hyperlinks between the Web documents (thereby capturing the relationships between the requested documents) with the clickstream (the sequence in which a user requests the documents) and the properties of the documents (position in search result, HTTP status code).

The goal of the model is to help answer numerous questions regarding a user's navigational behaviour during Web search, e. g.: Which link result position is likely to contain the answer a user is looking for? Is a user likely to explore the Web site of a

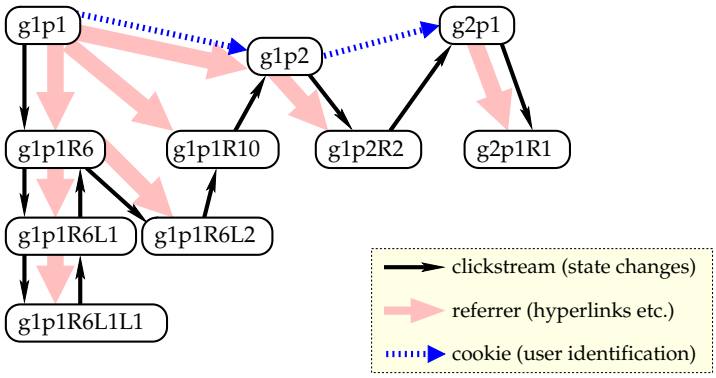


Figure 4.2: Logical relations (referrer, cookies) and actual click sequence as conducted by the user in a sample search session.

top-ranked result click further than a subsequent one? Do protections against "deep links" from search engines affect user search behaviour? Moreover, this model can be used as a refinement for the interdomain traffic estimation technique of Chang et al. [CJMW05].

4.6.1 States and transitions

Each *state* in our Markov model includes important aspects of the users' navigational behaviour. It retains the following information:

Index of search query in session: g_x

Index of result page: p_y

Position of result click: R_k, if it can be established

Furthermore, as a user may visit additional pages between result clicks, we capture the tree structure of such requests by keeping an index for the tree depth L_i and the number of sons for each level of the tree L_z. Moreover, a state may have additional attributes for capturing whether the page was reached via a different HTTP status code than "`200 OK`".

Each state captures the logical relationship of the requested page to the query that directly or indirectly made the user access this page. As the user clicks on pages, he navigates through the state space, and each click on a hyperlink corresponds to a state change. (Note, however, that requests served directly out of the client cache can only be inferred if they do not result in an `If-Modified-Since` request.) In effect, when viewing the clickstream as a set of events over time, the current state represents the user's (presumed) navigational position in the graph of hyperlinked documents accessed during the search session.

Let us consider an example of a search session where the user searches for information on the soccer world championship in 2006. The states in our Markov model are

given in parentheses; the entire search is depicted in Fig. 4.2. At first, the user might submit the query "soccer" to the Google search engine ($g_1 p_1$), and explore the sixth result link on page 1 ($g_1 p_1 R_6$), "soccer international root page", in a new browser window. On this Web site, he explores, e. g., the link for "German version" ($g_1 p_1 R_6 L_1$), where he follows yet another link ($g_1 p_1 R_6 L_1 L_1$). He finds that this link does not contain what he was looking for, thus presses the back button twice, which results in two `If-modified-since` requests ($g_1 p_1 R_6 L_1$, $g_1 p_1 R_6$), and clicks on the "English version" page instead ($g_1 p_1 R_6 L_2$). Still unsatisfied, he goes back to the initial search result list ($g_1 p_1$) and explores the tenth link: "soccer-sites.com" ($g_1 p_1 R_{10}$). As this site does not contain the desired information either, he takes a look at the next set of results from Google ($g_1 p_2$), where he clicks on the second result link ($g_1 p_2 R_2$) "ussoccer.com". This is still not a site about the FIFA World Cup in Germany, so he refines his original search by typing "Soccer world cup" in yet another browser window ($g_2 p_1$). Note that the usage of another browser window has the effect that we cannot establish a logical connection between the two clicks via the referrer field; rather, we have to rely on the PREF cookie. In this case, the first result ($g_2 p_1 R_1$) points to "The official site for the 2006 FIFA World Cup Germany", which the user clicks on. The thick pink lines in the background of Fig. 4.2 show the relationships (i. e., hyperlinks) between the Web pages, whereas the thin black arrows depict the actual user clickstream. The dashed blue arrows indicated queries whose affiliation cannot be inferred through HTTP referrers, but only via the PREF cookie.

We distinguish three types of states:

Query result pages have the form $g_x p_y$ indicating that this is the y-th result page of the x-th query in the session.

Result clicks have the form $g_x p_y R_\gamma$, where $g_x p_y$ identifies the query and γ is the result position.

Other clicks have the form πL_z where π is the state of hyperlinking document Π that originates the click, and z is the index of the clicked document in the vector of clicked hyperlinks originating from Π, ordered in time.

4.6.2 From HTTP logs to Markov states

For the construction of the Markov model, we focus on `text/html` objects, since each Web page has an HTML document as its skeleton. A brief analysis of HTTP `Content-Type`s reveals that the number of transferred objects in other potentially hyperlink-capable formats such as PDF or XML is insignificant at the time of this writing.

To capture the Web search clickstreams by means of a Markov model, we apply the following logic: For each new request ν, we first locate the session that it belongs to, using the method described in Section 4.4. Then we determine the state to be assigned to the request using the mechanism outlined in Fig. 4.3: First, we examine whether ν

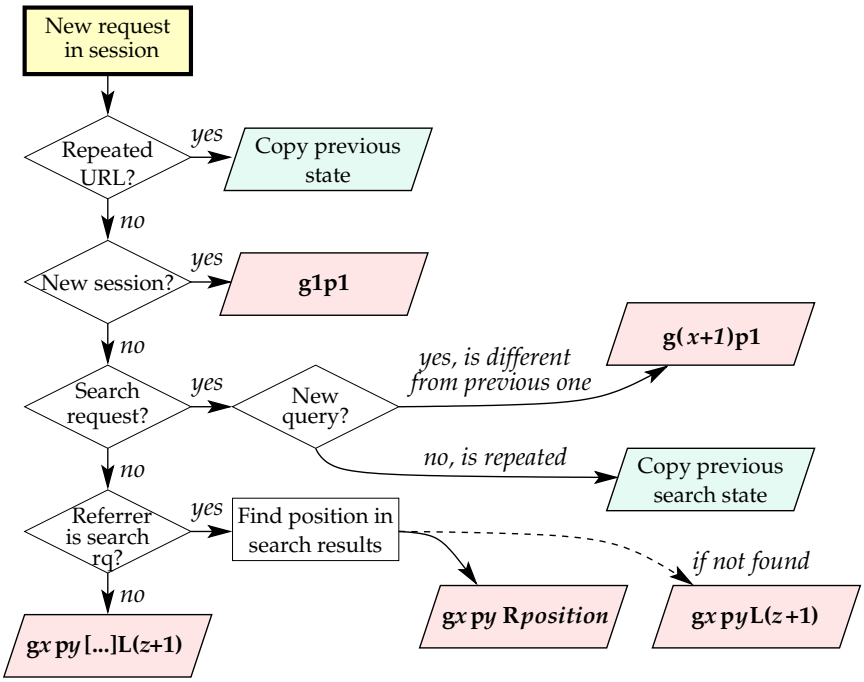


Figure 4.3: Determining the Markov state for a new request.

requested the same URL as a previous request ρ in the same session. If so, we simply assign ν the same state as ρ. Otherwise, we check whether ν is a search request. If so, we either increment the search index number from $g_x \ldots$ to $g_{(x+1)} \ldots$ (in the case of a new query), or we retain the old search number $g_x \ldots$ if the user repeats his query, e. g., he might request the next Google result page (which requires an adjustment of the page number from $g_x p_y$ to $g_x p_{(y+1)}$), or he might have entered the same search terms again. If the request ν is not a search request, we examine the request ρ that corresponds to the URL where the referrer of ν points to (this task is described in Section 4.4.2). Assume that ρ has state $g_x p_y[\ldots]$, and that previous requests already have been assigned the states $g_x p_y[\ldots]L_1$ to $g_x p_y[\ldots]L_z$. Then we assign ν the state $g_x p_y[\ldots]L_{z+1}$. An exception occurs if ρ is a search request: In this case, we determine the position of the URL for ν in the HTML code pertaining to ρ and thus determine its search rank. Here, the state we assign to ν is $g_x p_y R_{position}$, i. e., it depends on the position of ν in the list, but *not* on the number of child states of ρ.

Note that the user clicking on a single hyperlink may trigger the download of multiple `text/html` documents. For example, the URL that a hyperlink points to may result in a "`302 Found`" redirect (having `text/html` as `Content-Type`), which points to an HTML document consisting of multiple frames, each transferred as yet another individual `text/html` object. To avoid the instantiation of many futile intermediate states, we assign a request the same state as a previous request from the same session, if they are less than one second apart from each other and are linked via URL / referrer.

To keep Fig. 4.3 simple, this mechanism is omitted in the picture. In the case that we cannot capture the beginning of a session, a "new" session may start with, e. g., $g_1 p_3$ instead of the normal case $g_1 p_1$ shown in Fig. 4.3. This is due to the fact that we always calculate the page number ($g_{...} p_y$) from the search request's URL. We find only a small number of such exceptions in our data.

When identifying state transitions as indicated by the temporal evolution of the search clickstream, we find that the time sequence of the clickstream (thin black arrows) is likely to differ from the hyperlink graph as highlighted in Fig. 4.2 (thick light-coloured arrows). For example, by keeping the query result page in a separate window, the user can call $g_1 p_1 R_{10}$ without having to re-enter his query; thus a re-request of $g_1 p_1$ is not necessary. Note that the same page and therefore the same state can be reached multiple times, e. g., when the user presses the "back" button after retrieving page $g_1 p_1 R_6 L_1 L_1$.

4.6.3 A refinement for interdomain traffic analyses

Apart from describing those aspects of user behaviour that are immediately pertaining to Web searches, our model also can be used to refine the method that is described in [CJMW05]. In their work, Chang et al. draw on various publicly available data sources for estimating an interdomain traffic matrix. One of their data sources are public lists that contain frequently-used search terms in Web search. These lists are used to send queries to a popular search engine. The highest-ranked sites in the query result list are then identified, as they are considered to be major traffic sources due to an assumed search result popularity.

Using our method, it is possible to refine this method in a number of ways. First, our model yields weights on the popularity ratios for the differently-ranked entries in the search result page (i. e., the first search result is more often clicked upon than the second one, etc.). Second, as we have seen our (refined) model can tell on average how many more clicks a user spends on the website of the first, second, third etc. search result—and this number of clicks can be expressed in the number of bytes. Third, our approach of collecting HTTP traces can be used to identify popular sites in general, although this is, of course, not specific to our state model.

4.7 Model-Based Analysis

Recall that our state model does not only investigate the search queries, but also takes into account all subsequent clicks that are direct or indirect consequences of the users clicking on search results (followup clicks). In the following, we demonstrate the broad applicability of our model by highlighting some key findings that we made in our analyses of real-world data.

Using the data described in Section 4.5.1 with our state model, we are able to assign a state to 1,488,246 `text/html` requests with HTTP status code 200 and identify 1,336,418 "clicks" (search states). Thus **the share of search operations and their**

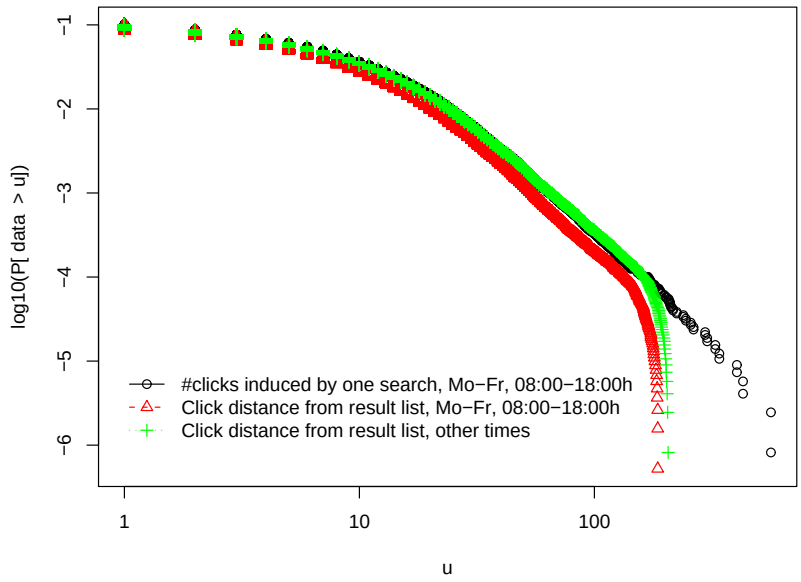


Figure 4.4: CCDF for number of clicks triggered by one search operation, and CCDFs for click distance between search page and visited document.

follow-up clicks amounts to least 6.8 percent of all transferred HTML documents. Note that this lower bound is strict.

If we analyse the distribution of the number of followup clicks for each individual search request (i. e., the "child states" for each $g_x p_y$), we see a mostly linearly falling slope on a CCDF plot (Fig. 4.4, circles). As was pointed out before in Section 3.4 and the glossary, this suggests consistency with a Zipfian distribution. In this respect, **search-triggered Web sessions thus do not seem to differ from Web sessions in general** [Bar01]. The same holds for the distribution of total the number of clicks per session (plot not shown).

Similar behaviour occurs in the distribution of the number of clicks between a document and the original search request, which we call *click distance*. In addition to this, Fig. 4.4 reveals that the users behave slightly different during working hours (red triangles) than during recreational times (green cross-hairs): **During their spare time, users are more likely to engage in "serendipity clicks" leading them away from the page they may have been originally looking for.**

Next, we compare the total number of clicks per session (black circles) vs. the distribution of click distances (red triangles) in the corresponding time intervals. We observe that these curves run in parallel for almost the entire range, except at the end. This suggests that **most users follow a rather linear approach during searching and browsing; i. e., they normally do not click on the "back" button and follow another link on the previous page,** etc. Only long sessions with many clicks seem to differ significantly in this respect (lower right of plot).

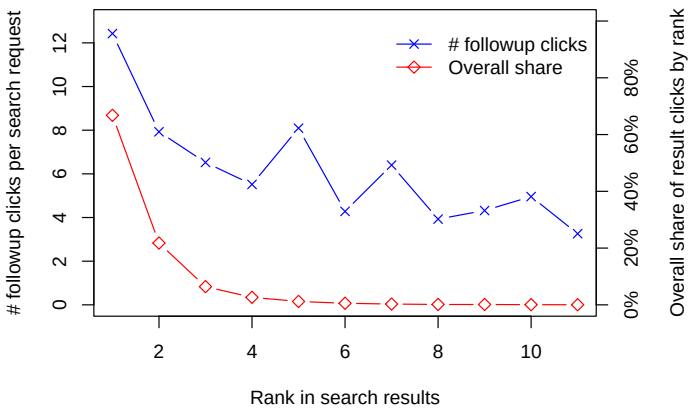


Figure 4.5: Rank in search result vs. willingness of the user to continue browsing from the result page.

Users are much more likely to re-formulate queries than to view the second result page. By comparing states $g_i p_1$ with $g_i p_2$ and $g_{i+1} p_1$, we observe that a user is about 37 times more likely to enter a new query than to look at the next result page.

If we look at the lozenges in Fig. 4.5 (bottom), we see that **most users request the first search result**—in fact, more than 60 % of the clicks on any ranked search result (i. e., excluding advertisements, search settings etc.) go to position 1. Note, however, that we witnessed automatic pre-fetching of the first search result in a number of cases.

When we consider the number of follow-up clicks to a search result as a measure for the result quality, we see that top-ranked search results indeed seem to be of higher quality than lower-ranked ones (Fig. 4.5, blue line at top with crosses). Yet, the presumed result quality difference between high ranks and low ranks is significantly lower than the difference in popularity (red line at bottom with lozenges). This suggests that the **page summaries in the result list probably are read by most users** before they click.

Let us finally analyse the impact of HTTP redirects that lead the user away from a search result: If a user enters any state $g_x p_y \{R, L\}_z$ via an HTTP redirect (not issued by the search engine), the average number of clicks that start from this document is only 0.12, as compared to the normal 6.5. This means that a user who clicks on a search result, but is redirected to a different page than the desired one, normally does not spend any time on that page. We conclude that **operators protecting their Website against "deep linking" from search engines repel many potential customers.** Ironically, their intent of trying to lure visitors into their site this way turns out to have just the opposite effect.

4.8 Summary and Outlook

In this chapter, we analysed Web search clickstreams. Our data is gathered by extracting the HTTP headers from packet level traffic, as well as the bodies of Web search result pages. We correlate both data sets to extract sequences of subsequently posed queries, and relate each query to its clicked result pages and follow-up clicks.

Based on the data gathered so far, we find that most queries consist of only one keyword and make little use of search operators, such as the plus, minus or tilde sign. Moreover, users issue on average four search queries per session, of which most consecutive ones are distinct. Relying on our Markov model that captures the logical relationships of the accessed Web pages, as well as navigational behaviour, we gain additional insights on users' Web search behaviour. Users are much more likely to re-formulate a query than to look at the second result page. This is consistent with the observation that the top-ranked results are much more attractive to a user, perhaps due to the reluctance to use the scrollbar. Moreover, judging from follow-up click behaviour, top-ranked results seem to be of higher quality than lower-ranked ones. "Serendipity browsing" seems to influence user search behaviour during recreational time periods. Finally, Web sites that are protected against "deep links" repel many visitors.

Our approach for gathering clickstreams is generic and not limited to Google, our example search engine. By gathering data from multiple search engines for the same user set during the same time period, comparisons of user behaviour (and, possibly, implications about search result qualities) across different search engines are possible. While it was our initial intention to investigate along this alley, we found to our greatest surprise that a sensible analysis of this kind was not possible with the data presented in this work—since the queries to Google appeared to overwhelmingly outnumber the queries posed at other search engines (e. g., Yahoo, MSN search, Lycos, Excite). Worse yet, those other search engines appeared to be used only during very short sessions, which makes a state model analysis futile. Another promising alley can be to perform timing-based analyses by considering the time a user spends in each state.

In a study that is related to our work presented in Chapter 3, Chang et al. identify hosts that serve Web sites containing frequently-accessed search terms from publicly available data [CJMW05]. They use this data to estimate interdomain traffic matrices, by estimating the traffic that was supposedly served by the servers hosting the pages with the most popular search terms. Combined with our methodology, this analysis can be refined further: For example, one can take into account not only the popularity of the query terms, but also the actual result clicks, or one can estimate the number of follow-up clicks on the same server. Moreover, our method allows to consider all kinds of search engines, including very popular ones. It is not restricted to publicly available query term popularity lists, which are a scarce quantity.

5 Memory-Efficient Traffic Statistics

The previous chapter introduced a methodology for estimating traffic demands on a global scale. Dynamic routing and traffic engineering, which is the focus of this thesis, indeed needs to draw on data sources that provide similar traffic information. Yet, the methods presented in Chapter 3 have some limitations, not only regarding the scope, but also regarding the time scale. Moreover, it requires data sources such as CDN log files which not many institutions, even large ISPs, may have access to. Furthermore, the method comprises centralised aspects regarding the collection of data, and suffers from some limitations regarding the timescale on which we can expect to obtain meaningful results.

A dynamic routing or traffic engineering mechanism, however, ideally works in a decentralised manner. This also comprises the task of collecting the data which the routing or TE decisions are based upon—and it needs very frequent updates on the traffic conditions, i. e., on a time scale of one minute or, preferably, less.

A natural first step towards a distributed traffic data collection is to let the routers count the oncoming traffic packets, using, e. g., NetFlow [Cis] or similar mechanisms. However, NETFLOW is known to negatively affect the performance of a router. This is a reason why many operators refrain from enabling it in their networks. One possibility to alleviate the NetFlow impact can be to use packet sampling methods [SF02]. On the other hand, emerging trends such as load-adaptive packet classifications and adaptive firewalls [BK05] require constant monitoring of hit rates in the packet classification data structures of a networking device. This information, too, can be used as a data source for dynamic routing and traffic engineering.

In this chapter, we present an algorithm for efficiently gathering statistics about the hit frequencies on the nodes of a search tree in a packet processing system, under limiting space constraints. The Expand and Collapse (EaC) algorithm is a heuristic that periodically adjusts the set of those search tree nodes at which statistics are gathered, in order to use the limited space available to collect statistics in preference from the currently most frequently visited nodes in the search tree. We prove convergence and good node-hit coverage of the algorithm and validate its performance on a set of simulated data.

The information collected can be useful for a variety of reasons, such as inferring traffic properties for dynamic routing, discovering failures and attacks, dynamically optimising a tree-based packet classification method for locality patterns in the oncoming traffic, or other optimisations related to the tree on which our method is used.

5.1 Introduction

search tree

packet classification

A common task in packet processing systems is a lookup or search over a database organised into some form of a search tree. Typically executed per every packet, it must be carried out within a tight time budget. Examples are a next-hop lookup, where a packet destination address is compared to a table of address prefixes using a longest-prefix match, or *packet classification*, where multiple fields in the packet header are compared against rules in the classification table, in order to determine the applying rule with the highest priority.

Efficient implementation of the longest-prefix match [NK98, DBCP97, WVTP97] and rule-based classification [FM00, GM99, SBVW03, KKV+03] has been a subject of extensive research. Often, a search tree is built over a pre-processed prefix-table or rule-set, to achieve a favourable tradeoff between size and search speed. For example, the HiCuts [GM99] and HyperCuts [SBVW03] classification methods construct search trees by a heuristic that cuts the search space into subspaces containing an approximately equal amount of rules; HiPac [FM00] constructs several interlinked search trees.

Generally, few assumptions or observations are made about the workload patterns of the search keys—the methods are typically optimised for the worst case scenario, minimising the depth of the search tree. However, neither the packet flows are distributed uniformly over the address- or rule-space, nor is the popularity of flows in terms of packet count uniform [FW97, WDF+05]. Clearly, such non-uniform distribution of workload is going to lead to massive inequalities in the number of packets traversing different paths of the search trees, as some paths will be traversed much more frequently than others.

To our best knowledge, little work has been so far dedicated to the ability to monitor, at run-time, the hit rates per different paths traversing the search tree structure. This information can be useful in a number of ways: for inferring traffic properties, determining the most potent traffic flows, discovering failures and attacks, or dynamically optimising the search method itself for locality patterns in the oncoming traffic, e. g., by providing shortcuts in the tree [BK05, KS06].

In this chapter, we study statistics gathering from a tree-search method on a high-speed architecture with a complex memory hierarchy, such as a network processor [JK03]. We aim to efficiently capture how traffic is distributed over the search tree and to determine the current frequently traversed paths.

5.2 Related Work

Gathering statistics about current network traffic has become a standard task in both research and in operational networking environments [IDGM01, MHH+03, EKMV04]. Usage of such statistics ranges from analysis of flow dynamics on wide-area networks [FW97, WDF+05] over rules how traffic can be engineered [PTD04, FGL+00a] to traffic-adaptive methods within the network node [SRS99, KB02].

The principal lessons can be summarised as that many quantities characterising network performance have long-tail probability distributions, which may have a dramatic effect upon performance [FW97, WDF+05]. Thus, a possible engineering principle is to select a small set of objects (packet flows) that account for a large fraction of the overall traffic (due to the long-tail distribution), to be treated differently so as to achieve a specific performance objective. However, the object dynamics are volatile and the particular objects (packet flows) need to be reselected often [PTD04].

The counter management algorithm in [RV03] allows to maintain a large number of counters at line rate. We take an orthogonal approach, by restricting the number of counters to only those that convey useful information, to further limit the overhead.

5.3 Efficient Gathering of Statistics

We assume that packet information is processed by performing a tree search operation using some packet data as the search key—more precisely, that the search starts from the root of the tree and continues downwards, without any backtracking, as in, e. g., HyperCuts [SBVW03] (Fig. 5.1). Our goal is to determine the frequently traversed tree paths and to measure the hit count on these paths.

5.3.1 Hardware context

network processor

We assume a programmable networking system, such as a network processor (short: NP) [JK03], to be the target device for the method's implementation. NPs typically consist of a control processor, multiple forwarding engines and a hierarchy of memories, differing in memory access latency, size and cost. Typically, the memory accesses are a key bottleneck in terms of performance. Fast memory is scarce and it is the use of this resource that we aim to optimise in this work.

data plane

control plane

The tasks executed on a NP belong either to the *data plane*, i. e., tasks that are processed per every packet, generally carried out at the forwarding engines, or to the *control plane*, i. e., not-so-frequent, more computationally intensive tasks carried out on the control processor. We assume the tree search method to belong among the data plane tasks, as well as the counting of hits along the nodes of the search tree. The algorithm to select the monitored nodes can run on the control plane, as the selection process may be relatively complex and is not required to happen on a per-packet timescale.

Gathering of hit statistics should not impose a large overhead on top of the search algorithm. We thus need to use fast memory for counting the accesses to the nodes. For example, on the Intel® IXP 2400 NP, there are 8×640 words of high-speed local memory available [JK03], yet the rule databases containing several thousands of rules can range from 10 k to 100 k words [GM99, SBVW03]. Furthermore, if the search is performed on multiple engines in parallel, simultaneous memory accesses must be dealt with. Thus, reducing the counting overhead is important to prevent potentially expensive access conflicts.

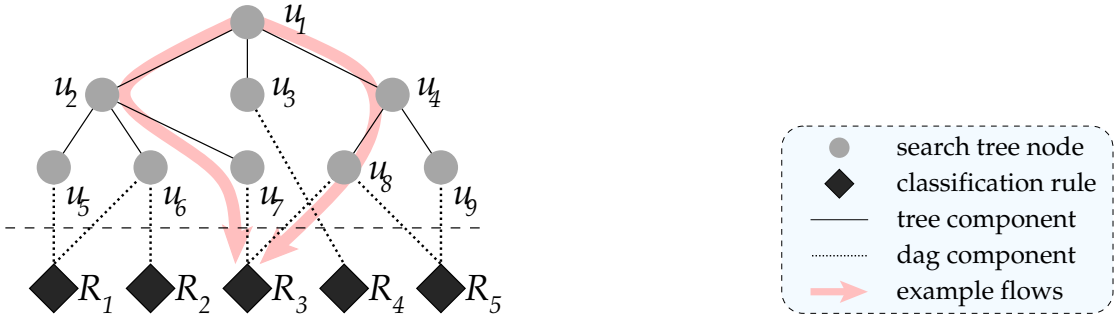


Figure 5.1: Small example packet classification search tree and corresponding classification rules. Note that, for example, rule R_3 can be reached both via the path (u_1, u_2, u_6), as well as (u_1, u_4, u_8).

5.3.2 Counting strategies

It is unwise to maintain a counter for each node of the search tree, as it is desirable to reduce the number of counters located in the fast memory, although other counters could be placed into slower but cheaper types of memory, as in [RV03]. A straightforward approach is to maintain counters for the tree leaves only. This allows to reconstruct hit counts up through the nodes in the tree by summing up hits of child nodes. However, the number of leaves in a typical classification search tree can be quite large (a tree of a uniform node degree 4 and depth 10, which is quite realistic [GM99, SBVW03], has about 1 million leaves). As most of the leaves most likely will be hit infrequently, monitoring all of them would waste too much memory on "uninteresting" counters.

In classification trees [GM99, SBVW03], many rules span different leaves of the tree (see Fig. 5.1). We thus might keep a hit counter for each rule, yet this approach wastes a lot of counters for rules that are accessed only rarely. Once determined, the counters for these infrequently-hit rules can be shifted to slower memory, but we would still need a vast amount of fast memory in the initial phase: The number of rules in today's ACLs is in the order of thousands [GM99, SBVW03, KKV+03], but is expected to grow fast with proliferation of mechanisms like VPNs or traffic engineering. Furthermore, by knowing how often each rule is hit, we cannot precisely infer how traffic traverses the tree. This is due to the fact that one rule may be reached via different paths (as in Fig. 5.1), each of which can have vast differences in access probabilities.

The method presented in this work collects data preferably at the nodes that attract a large number of hits. To monitor the heavily-hit nodes, we must determine the position of these nodes, but this in turn would require earlier monitoring of the tree structure. To address this chicken-and-egg problem, we iteratively find out and adjust which nodes are heavily-hit. In Section 5.6.3, we prove that our algorithm provides the precise hit count for all nodes that are hit at least $1/x \cdot \#\textit{search operations}$ times, if given $x \cdot \max\{\textit{height of tree}\}$ counters.

5.4 Notations and Definitions

In the case of the HiCuts [GM99] and HyperCuts [SBVW03] algorithms, there are three layers in the search tree structure:

1. the tree with the search nodes, each representing a set of cuts,
2. lists of rules attached to nodes, and
3. the rules themselves (see Fig. 5.1).

For our purposes, we perceive the lists of rules (2) as leaves of the tree. Coupling the rules (3) with the search tree would turn the tree into a directed acyclic graph, because one rule may be reached through different paths in the tree, as in Fig. 5.1. When referring to a *tree*, we however mean the part without the rules (i. e., without (3)). **tree**

In the remainder of this chapter, variables and symbols have the following meaning:

u_i	node i
u_0	root node
$\mathrm{f}(u_i)$	hit count (number of hits) on node i
t_k	time interval k
$\rightarrow_{(t_k)}$	transition to next time interval t_k
$\chi_{(t_k)}$	some "χ" during time interval t_k (e. g.: $\mathrm{f}_{(t_4)}$ = hit count during interval t_4)
$\mathrm{f}_{(t_k)}(u_0)$	hit count on root, i. e., packets processed by the system during t_k
$u_i \supset u_j$	node u_i is an *ancestor* of node u_j in the search tree
$u_i \dot{\supset} u_j$	node i is a *direct parent* of node j
$\mathfrak{M}u_i$	node i is being monitored (predicate)
$\mathfrak{F}u_i$	node i can be followed (predicate)
$\mathfrak{H}u_i$	node i is heavily-hit (predicate)
M	set of monitored nodes
m	number of monitored nodes (i. e., $\lvert M\rvert$)
h	maximum height of search tree
n	number of nodes in the search tree
$\lvert\lvert u_i\rvert\rvert$	distance between u_i and the root of the tree
O	minimum outdegree of the tree

Definition 5.4.1. *A node u is said to be **heavily-hit** relative to a set of nodes $\{u_1, \ldots, u_n\}$ iff*[1], *assuming that $f(u_1) \geq f(u_2) \geq \cdots \geq f(u_n)$ holds, $f(u) \geq f(u_\varrho)$, for some fixed $\varrho \in [1, n]$.* **heavily-hit**

[1] As usual, the word *iff* is shorthand for the logical expression "if and only if".

We use the metric of hit counts for determining whether a node should be monitored or not. Other metrics may be possible; however, the algorithmic rules in Section 5.5 use the assumption that the values generated by the metric are monotonically non-increasing along any path down the tree, and thus $(u \subset v \wedge \mathfrak{H}u) \rightarrow \mathfrak{H}v$.

Input traffic patterns undergo significant changes over time. We divide time into intervals $t_1, t_2, \ldots$, of equal length, and associate a node's property of being heavily hit ($\mathfrak{H}$) to a specific time interval.

5.5 The Expand and Collapse (EaC) Algorithm

Our method for efficiently gathering statistics under stringent memory constraints is divided into three parts:

1. The actual **statistics gathering** happens during the classification of a packet. Updating of the counters must be integrated into the traversal of the tree during the search. Due to accessing counters, this part of the algorithm needs to operate using fast memory, as the per-packet processing time is potentially prolonged.

2. **Selection of monitored nodes**, i. e., the nodes for which we keep counters, is performed after each time interval of a pre-defined length. The selection is performed based on the data read from the counters and may be done off-line. This part constitutes the actual *EaC algorithm*.

3. **Bounds computation for other tree nodes**, i. e., inferring from our knowledge about the monitored nodes to other nodes of the tree, which can also be performed off-line. This part can be implemented as an on-demand API for the application that requires the statistics being gathered, e. g., a tree optimisation algorithm.

5.5.1 Idea

As we are looking for the heavily-hit nodes, we start monitoring nodes near the root of the tree first, and then proceed towards the leaves selectively at the heavily-hit nodes. Whenever the traffic characteristics change, we gradually adjust the monitored set to fit the new distribution of node hits—i. e., we refrain from watching nodes no longer heavily-hit and instead monitor nodes that have recently become heavily-hit.

To keep track of which nodes are monitored, we augment all the nodes of the monitored tree by a flag `monitored`. Formally, for a node u we indicate monitoring by a predicate $\mathfrak{M}u$. Furthermore, the nodes being monitored carry the following additional fields:

monitored nodes

- A *counter* that counts all the search accesses passing through this node;
- A flag "`follow`", indicated by a predicate $\mathfrak{F}u$. The purpose of this will be explained further below.

follow flag

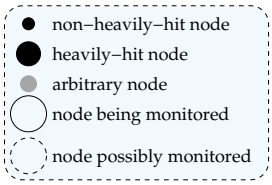


Figure 5.2: Legend for the rule figures 5.3, 5.4, 5.5, 5.6.

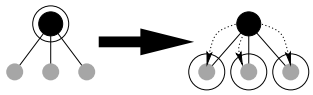


Figure 5.3: Expand-to-children rule: If a node is heavily-hit, we start monitoring its children.

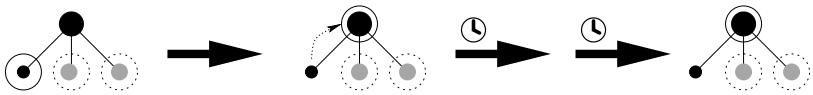


Figure 5.4: *Collapse* rule, with the oscillation prevention rule: If a monitored node is not heavily-hit, we stop monitoring it and monitor its parent instead. The oscillation prevention stops us from re-descending to the non-heavily-hit node for a configured time period.

After each monitoring time interval t_i, the counters of the monitored nodes are read (i. e., the $\mathrm{f}_{(t_i)}(u)$ values for each monitored node u) and reset, and the monitoring attributes of the nodes are changed, applying the algorithmic rules that are described below. Note that the old values of $\mathrm{f}_{(t_i)}(u)$ can be stored in slow memory once the time interval t_i has elapsed.

5.5.2 The EaC algorithmic rules

After each time interval, we adjust the choice of nodes to monitor, according to the following rules (a legend for the rule description figures is given in Fig. 5.2):

Expand to children

If a node is monitored and found to be heavily-hit, then monitor all of its children. Stop monitoring that node itself (see Fig. 5.3). Formally, **expand rule**

$$\mathfrak{M}u_p \wedge \mathfrak{H}u_p \rightarrow_{(t_{i+1})} \neg\mathfrak{M}u_p \wedge (\forall u_c \dot{\subset} u_p : \mathfrak{M}u_c)$$

Collapse to parent

If a node is monitored and found to be non-heavily-hit, stop monitoring it, and instead start monitoring its parent again (see the first step in Fig. 5.4). If this rule is applied to n nodes that are siblings, then we obviously free $n-1$ counters. Formally, if $u_p \dot{\supset} u_c$: **collapse rule**

$$\mathfrak{M}u_c \wedge \neg\mathfrak{H}u_c \rightarrow_{(t_{i+1})} \neg\mathfrak{M}u_c \wedge \mathfrak{M}u_p$$

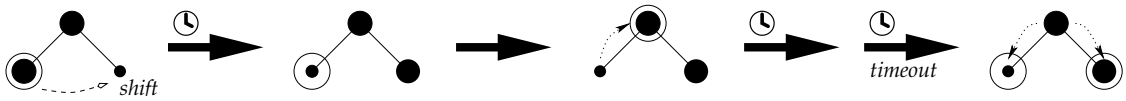


Figure 5.5: The `follow` flag timeout. The `follow` flag is reset after a configurable timeout, to allow for finding newly heavily-hit children, to whom the *collapse* rule has previously been applied.

Prevent oscillation

oscillation prevention rule

Suppose that a node is heavily-hit, but actually none of its children is heavily-hit. In this case, the children are first examined using the *Expand* rule. Since all the children are non-heavily-hit, the *Collapse* rule will be applied to each during the next EaC run; thus in the next time period, it is the parent node again that will be monitored. Obviously, such a scenario can easily lead to infinite oscillations. Thus, upon applying the *Collapse* rule, we clear the `follow` flag (set true by default) for the parent that is being monitored again:

$$\mathfrak{M}u_c \wedge \neg\mathfrak{H}u_c \rightarrow_{(t_{i+1})} \neg\mathfrak{M}u_c \wedge \mathfrak{M}u_p \wedge \neg\mathfrak{F}u_p \wedge \mathfrak{F}u_c$$

We amend the *Expand* rule such that it is only applied to nodes that have the `follow` flag set:

$$\mathfrak{M}u_p \wedge \mathfrak{H}u_p \wedge \mathfrak{F}u_p \rightarrow_{(t_{i+1})} \neg\mathfrak{M}u_p \wedge (\forall u_c \dot{\subset} u_p : \mathfrak{M}u_c)$$

The `follow` flag is re-set in collapsed children, as we might have to re-examine them at a later time. See Fig. 5.4 for a graphical representation of the *Collapse* rule combined with the oscillation prevention rule.

After a configurable number of time intervals, we re-set the `follow` flag in the parent, allowing to apply the *Expand* rule again (see Fig. 5.5). Formally,

$$\cdots \rightarrow_{(t_i)} \neg\mathfrak{F}u \rightarrow \cdots \rightarrow_{(t_{i+\text{timeout}})} \mathfrak{F}u$$

This is to allow for finding children that have recently become heavily-hit due to changes in the network traffic patterns, and to whom the *Collapse* rule had previously been applied.

Remove intermediate monitored nodes (optional rule)

Optionally, a further optimisation rule may be applied. This rule can boost the efficiency of the monitoring process by eliminating further redundancies in the choice of the monitored set, in that it trades measurement precision for a reduced number of counters. It depends on the user's preferences to decide about the right tradeoff for the particular application, but the basic algorithm functions just as well without these optimisations.

remove intermediate nodes

Due to the *Collapse* rule, it may happen that we monitor nodes which also have a few heavily-hit descendant nodes, these being monitored as well. We can infer from the existence of these heavily-hit descendant nodes that the intermediate nodes *must*

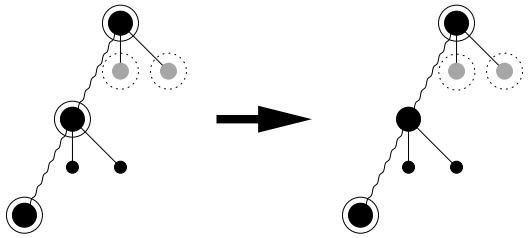


Figure 5.6: Remove intermediate monitored nodes: Since the parents of a monitored heavily-hit node must be heavily-hit themselves, there is little gain in monitoring more parents than only one at the top.

be heavily-hit as well, so there is no real need to monitor them (see Fig. 5.6 on the next page), except if we are interested in obtaining their *exact* hit number rather than a bounded estimate. To save counter positions, we therefore may choose to cease monitoring these intermediate nodes. However, this rule may prolong discovery of recently emerged heavy-hitters among the removed node's children.

$$
\begin{array}{rccccccc}
 & \mathfrak{M}u_1 & \wedge & \mathfrak{M}u_2 & \wedge & \mathfrak{M}u_3 & \wedge & \\
 & \mathfrak{H}u_1 & \wedge & \mathfrak{H}u_2 & \wedge & \mathfrak{H}u_3 & \wedge & \\
 & (u_1 & \supset & u_2 & \supset & u_3) & \wedge & \neg\mathfrak{F}u_2 \\
\rightarrow_{(t_i+1)} & \mathfrak{M}u_1 & \wedge & \neg\mathfrak{M}u_2 & \wedge & \mathfrak{M}u_3 & &
\end{array}
$$

The rule is explained graphically in Fig. 5.6. Note that we always need to keep watching some uppermost node u_3 (i. e., at least the root node u_0), since otherwise we will never notice any sudden traffic increases at or above u_3.

5.5.3 EaC algorithm iterations

The algorithmic rules for selecting the monitored nodes allow the algorithm to work with a given number of counters efficiently. Each algorithm iteration consists of the following steps:

1. Order the currently monitored nodes by their respective hit counts.

2. Start at the top of the list (i. e., the heavily-hit nodes) and apply the rule that requires more counters—i. e., the *Expand* rule.

3. Free the additional memory required for the new counters by applying the *Collapse* rule as often as needed, starting at the bottom of the list.

4. Goto step 2 and repeat, until the *Expand* and the *Collapse* parts meet (i. e., the expansion cannot use any more collapse rules without having to remove nodes that it just expanded during the same iteration). If an *Expand* operation fails because we cannot free enough memory positions, then perform a rollback of this operation and all operations related to it, and terminate.

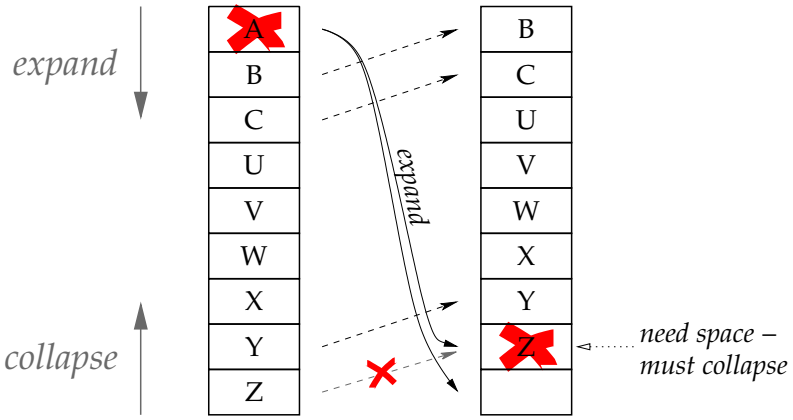


Figure 5.7: By applying the *Expand* and *Collapse* rules, the algorithm determines a new boundary between heavily-hit (new entries at the top) and non-heavily-hit nodes (formerly heavily-hit nodes are removed at the bottom), until no further expansion is possible.

EaC!iteration

An example for the resulting change to the list of monitored nodes is depicted in Fig. 5.7. We call the execution of this loop until its termination one *EaC iteration*. The next EaC iteration will take place after the next monitoring time interval has elapsed. An example of monitoring a very small tree with several EaC iterations is shown in Fig. 5.11.

The boundary between heavily-hit and non-heavily-hit nodes is thus not fixed, but rather re-calculated by the algorithm during each iteration, based upon the hit counts and the number of counters available.

collapsing threshold

If interested in obtaining the precise hit count on every node hit at least by a $\frac{1}{x}$ fraction of traffic, the EaC iteration must stop at a point where we would have to remove the counter from such a node, i. e., we refrain from applying the *Collapse* rule to nodes that are hit more than $\frac{1}{x}\,f(0)$ times (the *collapsing threshold*). See Section 5.6.3 for further discussion of this particular task.

5.5.4 Non-monitored nodes

The purpose of the EaC algorithm is to select the monitored nodes. We can, however, obtain hit counts on non-monitored nodes, too. The *precise* node hit count can be deduced by summing up the hit counts of its children, if known. As with the precise number of hit counts (Fig. 5.8), this can be done recursively. See theorem 5.6.4 for more details.

As no node can have more hits than its parent, we can use the precise node hit count as an *upper bound* for all of its descendants (Fig. 5.9).

If we are to remove some of the intermediate monitored nodes (Section 5.5.2), we can end up in a situation where not all children of a node are covered by counters, and where we thus cannot reconstruct the precise number of hits on this node. However, even with an incomplete coverage we can at least provide a lower bound: If some of

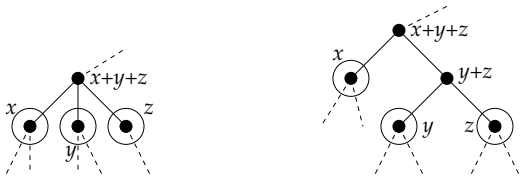


Figure 5.8: Summing up children: The number of hits on a node can be derived by summing up the counters of all of its children (left). The summation may be applied recursively to all nodes of the tree (right).

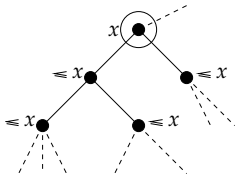


Figure 5.9: Upper bounds: The number of hits on a node is an upper bound for any of its children, and thus recursively for any of its descendants.

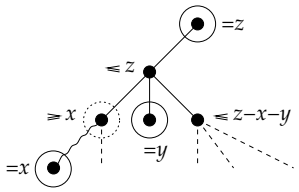


Figure 5.10: Obtaining better upper bounds using information from siblings: An upper bound can be improved by subtracting the sum of lower bounds for all the siblings.

the node's children receive x hits in total, their parent must receive at least x hits too. We can recursively sum up the lower bounds (see Fig. 5.8). We obtain a lower bound for *any* ancestor of a monitored node.

A bound for a node can be tightened by including hit counts (or lower bounds, respectively) for its siblings: For example, if a node's parent is hit by at most z hits and its siblings are hit $\geq x + y + \ldots$ times, then the node can be hit at most $z - x - y - \ldots$ times (see Fig. 5.10). We may therefore be able to compute both the upper and the lower bound for some intermediate nodes.

5.6 Algorithm Properties

Before we give an experimental evaluation of the algorithm's performance, we first prove that it converges, and that it can provide us with the exact number of hit counts for a certain class of certain nodes. We also give a lower bound for the number of these nodes in relation to the tree size and the number of available counters.

5.6.1 Convergence

In this subsection we prove that the EaC algorithm converges to a stable choice of monitoring nodes. Since convergence is difficult to define in the case of a moving target (i. e., ever-changing traffic patterns in the real world), we analyse the case of a tree whose access patterns remain constant.

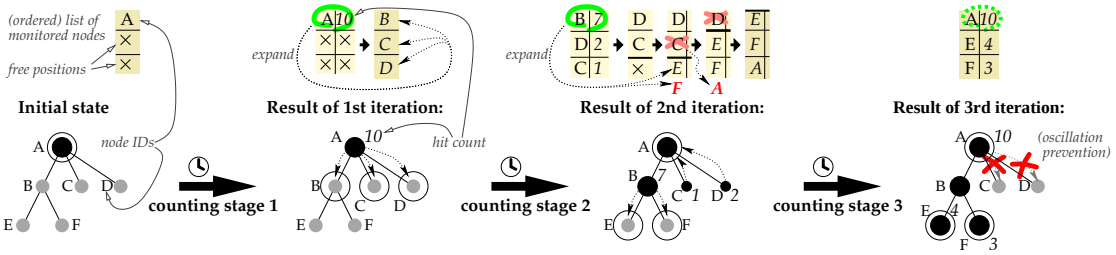


Figure 5.11: EaC example iterations with 3 available counters. After the first hit-counting stage, EaC is run on the tree. It expands node A; i. e., nodes B, C, D get monitored. The second hit-counting stage reveals that node B (7 hits) should be expanded, which requires to collapse node C (1 hit). However, this implies monitoring A again, so D (2 hits) needs to be collapsed also. After the fourth iteration, node A has more hits (10) than E (4) and F (3) thus should be expanded; however, this is not allowed if oscillation prevention is in effect.

Theorem 5.6.1. *In the case of constant hit patterns and a large enough timeout value, the EaC algorithm converges.*

Proof by induction over tree potential[2]**:** We define the potential of the tree and then show that it is strictly monotonically decreasing over the EaC iterations.

Definition 5.6.1. *We define the potential $\Phi(u)$ of a node u as*

$$\Phi(u) := \begin{cases} 2 & \Leftrightarrow u \text{ has never been monitored yet} \\ 1 & \Leftrightarrow u \text{ is currently being monitored} \\ 0 & \Leftrightarrow u \text{ is not monitored any longer} \end{cases}$$

Moreover, we define the potential $\Phi(\mathcal{N})$ of a set of nodes $\mathcal{N}$ to be the sum of the potentials of the nodes:

$$\Phi(\mathcal{N}) := \sum_{u \in \mathcal{N}} \Phi(u)$$

This means in particular that the potential $\Phi(T)$ for the entire tree T is the sum of the potentials of all tree nodes.

Lemma 5.6.2.
(a) The initial potential $\Phi_{(t_0)}(T)$ of any tree T is finite.
(b) The potential of a tree is always non-negative, i. e., $\forall t_i : \Phi_{(t_i)}(T) \geq 0$.

Proof of lemma 5.6.2**:** Both statements are obvious, since (a) we operate on finite trees, and (b) the potential is a sum of non-negative numbers. □

Lemma 5.6.3. *The potential $\Phi(T)$ of any tree T with constant hit patterns is strictly monotonically decreasing when applying the EaC algorithm to the tree.*

[2]Proof method as in [CLRS01]. For a brief explanation of this method, see [Wik07c].

Proof of lemma 5.6.3**:** The potential of the tree only changes at nodes whose monitoring state is changed. This is only possible if these nodes have been affected by one of the rules described in section 5.5.

Let $\Delta\Phi(x) := \Phi_{(t_i)}(x) - \Phi_{(t_{i-1})}(x)$ be the difference in potential of some node x after an iteration $t_{i-1} \rightarrow t_i$. Following this definition, $\Delta\Phi(x) < 0$ means that the potential of x has decreased. Assume that during one iteration of the algorithm, one application of a single rule has affected the monitoring state of each node in $\mathcal{N} = \{u, v_1, \dots v_n\}$ with $\forall v_i : u \dot{\supset} v_i$. Now distinguish the following cases of rules being applied:

Expand: Monitoring of u has been expanded to monitor each v_i. Due to oscillation prevention, u can not have been monitored at an earlier stage—since then we would not have been allowed to expand u again. This means that none of the v_i had been previously monitored either. Thus $\Delta\Phi(\mathcal{N}) = \Delta\Phi(u) + \sum_{i=1}^{n} \Delta\Phi(v_i) = (0-1) + n \cdot (1-2) \underset{(\text{as } n \geq 2)}{\leq} -3$.

Collapse: Assume w.l.o.g. $n = 1$. As u must have been monitored before, we have initially $\Delta\Phi(\mathcal{N}) = \Delta\Phi(u) + \Delta\Phi(v_1) = (1-0) + (0-1) = 0$. However, the *collapse* rule is only applied if there is need to free a memory position. This need can only arise in two situations: (a) another node $\check{u} \dot{\supset} \{\check{v}_1, \dots\}$ has been expanded, or (b) another node $\hat{v} \dot{\subset} \hat{u}$ has been collapsed, which required freeing the memory position of v needed by $\hat{u}$. In case (b), the *collapse* rule is invoked recursively, but since we are dealing with a finite tree, this only may happen a finite number of times k. The recursion thus must have been triggered by one initial *Expand* rule (i. e., case (a)). In the end, we have a chain of rule applications, and the potential of all the nodes $\tilde{\mathcal{N}}$ affected by this rule chain is thus $\Delta\Phi(\tilde{\mathcal{N}}) = \Delta\Phi(\textit{expand}\text{ rule}) + (k+1) \cdot \Delta\Phi(\textit{collapse}\text{ rule}) \leq -3 + 0$.

Thus, no matter what rule has been applied, the tree potential $\Phi(T)$ is always reduced. □

To summarise, $\Phi(T)$ starts from a positive finite value, decreases strictly monotonically over the iterations of the EaC algorithm, but remains non-negative. This concludes our proof that the algorithm converges at some point. □

5.6.2 Precise hit count reconstruction

Theorem 5.6.4. *If v is a monitored node, we obtain the precise hit count on v and all its ancestors.*

Proof: Assume otherwise. Then there are nodes $u \supset v$ with $\neg\mathfrak{M}_{(t_k)}u \wedge \mathfrak{M}_{(t_k)}v$. Since $\mathfrak{M}_{(t_k)}v$, there must have been some previous t_i, $i < k$ such that $\mathfrak{M}_{(t_i)}u$. Node u can only have been ceased being monitored due to either the *Collapse* or the *Expand* rule. The former would result in $\neg\mathfrak{M}v$ **before** $\neg\mathfrak{M}u$ (since $v \subset u$ and thus $\mathrm{f}(v) \leq \mathrm{f}(u)$)—a contradiction. The latter must result in u being fully covered by its descendants—a contradiction to the assumption that we cannot obtain the precise hit count $\mathrm{f}_{(t_k)}(u)$. □

Note that the theorem implies that any path from the root to an arbitrary leaf passes at least one counter, and the monitored nodes thus form a cut across the tree structure.

5.6.3 Hit coverage

A common definition of a heavy-hitter object is one that receives at least $\frac{1}{x}$ of the total traffic [PTD04]. We now prove a relationship between x and the number of counters made available to EaC.

Assume that the search pattern remains constant over a number of iterations. Let $h := \max\{\textit{height of the tree}\}$. Then EaC converges to a state where it provides the precise hit count on all nodes hit by at least $\frac{1}{x} \cdot \mathrm{f}(0)$, using $x \cdot h = |M|$ counters.

Lemma 5.6.5. *The number of nodes that are hit at least $\frac{1}{x} \cdot \mathrm{f}(0)$ times, and that are deepest down in the tree, is at most x.*

Proof: If one node is hit by $\geq \frac{1}{x} \cdot \mathrm{f}(0)$ traffic, then its parent also must be hit by $\geq \frac{1}{x} \cdot \mathrm{f}(0)$ of the traffic. Since the nodes cover disjoint search space areas, there can be at most x such "deepest" nodes, each attaining $\geq \frac{1}{x} \cdot \mathrm{f}(0)$ hits. □

Lemma 5.6.6. *To obtain the precise hit count on all nodes that are hit $\geq \frac{1}{x} \cdot \mathrm{f}(0)$ times, we need to monitor at most $x \cdot h$ counters.*

Proof: Since we have $\leq x$ "deepest" $\frac{1}{x}$-hit counters, all additional nodes that we have to monitor must be parents of these x counters. This means that we have to monitor all nodes along the x paths from the root to each of these deepest $\frac{1}{x}$-hit nodes. The worst case is that these paths already separate immediately below the root, and that $\textit{height}(\text{deepest } \frac{1}{x}\text{-hit nodes}) = h$. This implies that we need to monitor $\leq x \cdot h$ nodes in total. □

Theorem 5.6.7. *If $x \cdot h$ monitors are available, then the EaC algorithm picks all of those nodes that are hit at least $\frac{1}{x}\,\mathrm{f}(0)$ times, if the hit patterns do not change until the algorithm has converged and if the* collapsing threshold *(5.5.3) is applied.*

Proof: Obviously, the *collapsing threshold* guarantees that we never cease monitoring $\frac{1}{x} \cdot \mathrm{f}(0)$-hits nodes once we have found them; we only may expand them and this way can detect possible $\frac{1}{x} \cdot \mathrm{f}(0)$-hit children.

EaC converges to a state where it has found *all* $\frac{1}{x} \cdot \mathrm{f}(0)$-hits nodes, because (a) during each iteration, it always expands those nodes that are hit most frequently, (b) we always obtain precise hit count on any node that is monitored or a parent of a monitored node, (c) EaC does converge. □

5.7 Performance Evaluation

In this section, we present some results from running the EaC algorithm within a simulator environment. We then analyse the results yielded by its choice of monitored nodes.

5.7.1 Evaluation set-up

For a number of reasons, it is not possible to pin down a "typical" packet classification tree. First of all, many different algorithms for packet classification exist (simple binary trees, HiCuts [GM99], HyperCuts [SBVW03], HiPac [FM00], and many more). These different algorithms naturally create different packet classification trees, even if the same set of input rules is given. Second, there is no "standard" rule set for packet classification. For example, the ruleset for a typical firewall focuses on port numbers and its rules focus only on a few source and destination networks. In contrast, the rules applied by a backbone router will be vastly different, since they will almost exclusively take destination prefixes into account; the rules for the access router of an MPLS-enabled network with quality-of-service routing decisions will have even other rules. From the fact that neither a standard packet classification algorithm, nor a typical packet classification rule base exist, obviously follows that a "typical" packet classification tree cannot exist either.

In order to evaluate the performance of the EaC algorithm, we chose to implement the algorithm within a generic simulation framework written in Java. Our framework allows EaC to be run on search trees of arbitrary shape, and generates random requests on them. At each node, the search requests follow a fixed user-defined probability distribution that changes from time to time. In our evaluation, we use artificially constructed trees, both uniformly constructed trees which we use to derive the general behaviour of algorithm in trees of different outdegrees, as well as random trees. The randomly constructed trees, however, are not completely random, but are constructed in a way such that they resemble trees produced by the HiCuts or HyperCuts algorithms.

OD
random tree

We use various uniform trees of fixed outdegree (OD) at each node and a fixed depth (d), as well as several randomly-built trees. The random trees were constructed as follows: At each node, we randomly determine if the node will have children (probability=0.75) or not; but at the same time, we ensure that all paths from the root to a leaf have a minimum length of 4 and a maximum length of 24. The number of children OD_u for a node u is determined by creating a uniformly distributed number $\in 2 \ldots 32$, which we then round down to the nearest power of two. The process is also stopped when the number of leaf nodes is equal to or greater some threshold; we use thresholds of 4096 and 65536. This is done to ease comparison of simulation results on trees that are different in shape but comparable in size. We create 9–20 instances for each tree shape and analyse the average values of the simulation runs in each equivalence class.

Since the EaC algorithm is designed to identify heavily-hit nodes in a tree, it is interesting to analyse the impact of different access patterns in the tree on the performance. Luckily, search operations in EaC-compatible search trees can be simulated in a simple fashion: We go through the tree from the root towards the leaves. At each node, we randomly pick a child to visit next according to some probability distribution. Each node has its own probability distribution for selecting child nodes that is kept constant over many lookup operations. We examine the effect of different probability distributions, i. e., uniformly, exponentially and Pareto distributed access probabilities (see below).

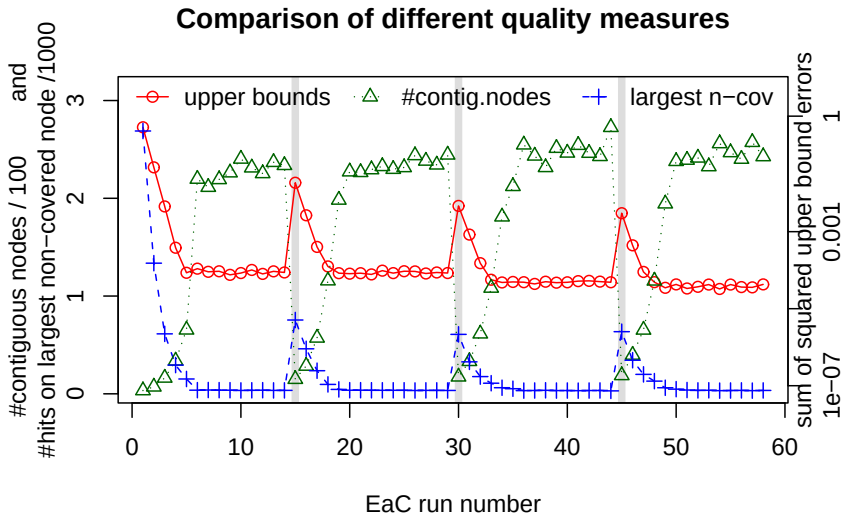


Figure 5.12: CDF showing EaC convergence. The algorithm converges rapidly, and the different quality measures behave consistently with each other.

After $5 \cdot 10^5$ requests have been issued, we run EaC on the tree to select a new set of monitors. This is followed by another $5 \cdot 10^5$ search requests, then another EaC run, etc. We chose a constant number of search requests instead of a random distribution in order to make results from independent EaC iterations easier comparable.

Every 15 monitoring intervals, the simulator imposes an entirely new hit pattern distribution at each node. This change is likely to render the previously-found set of monitored nodes useless, and is done in order to analyse EaC's reaction to pronounced dynamic changes to the oncoming hit patterns.

5.7.2 EaC performance measures

Before we let the EaC algorithm undergo a performance evaluation, we first have to define how we want to measure its performance. Since the EaC counter choices provide exact measures for monitored nodes and upper bounds for all other nodes of the tree, one aspect of the performance of the EaC algorithm is the accuracy of these bounds. As a measure, we define the (normalised) sum of *squared errors for the upper bounds* of each node in a tree T as

squared errors

$$\mathcal{E}(T) := \frac{\sum_{v \in T} (\mathrm{u}(v) - \mathrm{f}(v))^2}{|T| \cdot \mathrm{f}^2(0)}$$

Here, a better performance is indicated by a *smaller* value.

Another interesting aspect is EaC's performance in finding the "heavy-hitters" among the nodes. To this end, we determine the maximum number of hits on any node for which the algorithm cannot guarantee to yield the precise number of hits. We call this

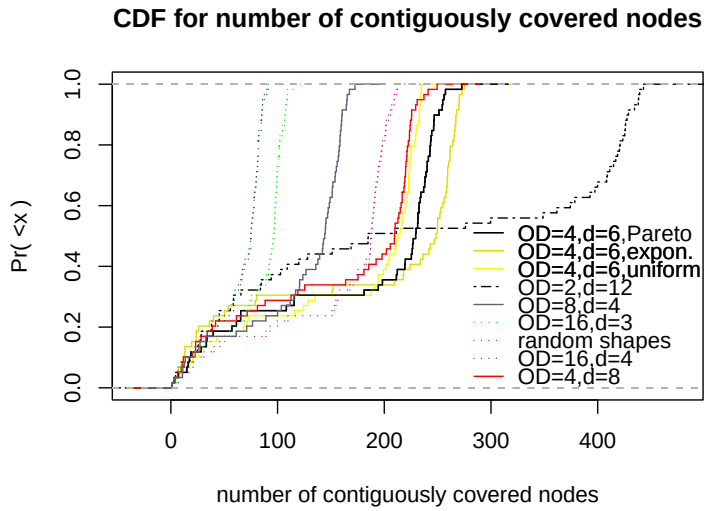


Figure 5.13: Influence of tree topology, size, and hit distribution.

the *largest non-covered node*. As with the error sums, better performance is indicated by *smaller* values of this measure.

largest non-covered node

Closely related is the number of nodes that are hit more often that the largest non-covered node. By definition, EaC yields their precise number of hits. We call this measure the *number of contiguously covered nodes*; better EaC performance is indicated by a *larger* number of this value.

contiguously covered nodes

5.7.3 Comparison of performance measures

Fig. 5.12 shows the average of simulation runs on 20 trees. Each tree is uniformly shaped ($OD = 4$, $d = 6$, 4096 leaves), whereas the individual trees have different Pareto-distributed hit patterns. We make 512 counters available to EaC, and use a uniformly distributed random variable from the interval $(0 \ldots 2)$ as the timeout value for the oscillation prevention rule. We will refer to this set-up as our *default setup* in the further analyses.

default setup

The (typical) plot in Fig. 5.12 on the previous page demonstrates that all four performance measures described above show consistent behaviour at the beginning and after each time the simulator installs new hit probabilities. In the following, we thus concentrate on only one performance measure. We pick the number of contiguously monitored nodes, since we believe this pattern to be less affected by the different tree shapes, which facilitates making comparisons.

5.7.4 Hit patterns

We now investigate the effect of the shape of the tree on EaC's performance. Fig. 5.13 shows some of the results for different groups of search trees of comparable sizes. To compare the effect of different hit distributions on a tree, we simulate different hit patterns on various trees while keeping our default values described above for all other parameters. A hit pattern is defined as follows: At each node, the probability to continue the search operation (during, e. g., packet classification) in either child 1 or child 2 or child 3 or (...), is (a) uniformly, (b) exponentially, (c) Pareto-distributed with $\alpha = 1.3$.

We notice that the hit distribution has a measurable, but not very pronounced effect on EaC's performance: The three lines for $OD = 4$, $d = 6$ are not far apart. It seems that a uniform hit pattern poses the worst case in this scenario (i. e., the line appears shifted towards the left) and exponential the best, Pareto lying in the middle. Due to the small impact of the hit pattern, we can restrict ourselves to investigating only one hit pattern. We chose to use Pareto-distributed hit patterns for this purpose, as flow popularities in the Internet typically are consistent with power laws [WDF+05].

5.7.5 Tree shape and tree size

By examining the other lines in Fig. 5.13, we see that the number of contiguously covered nodes decreases significantly as the branching factor (OD) increases: For $OD = 2$ we get the best coverage, even though the corresponding uniform binary tree has more nodes (8191) than, e. g., a tree with $OD = 16$ (4369 nodes) having the same number of leaves: The performance for the latter obviously is worse. Consistently, the randomised trees (average $OD = 7.1$) show a better performance than the trees with $OD = 8$.

If we increase the size of the tree while keeping the number of available counters at the same level, we intuitively expect the algorithm's performance to decrease. Indeed, this is the case if we keep the number of counters at 512 but increase the size of the tree: If we increase the size of the tree by a factor of 16, the performance is measurably reduced, although obviously not by a factor of 16. If, for example, we increase the tree depth for $OD = 4$ from $d = 6$ to $d = 8$, we see in Fig. 5.13 on the previous page that the corresponding line for the tree with the greater depth and thus the larger number of nodes is leaned more towards the left-hand side. The same holds if we compare the line for $OD = 16$, $d = 3$ to the line $OD = 16$, $d = 4$, which also corresponds to a 16-fold increase in the number of nodes.

5.7.6 Oscillation prevention rule

Next, we analyse the benefits of applying the oscillation prevention rule while keeping all other values at default. In Fig. 5.14, we analyse EaC oscillation prevention timeout values of 0 (i. e., no prevention), a fixed value of 2, and uniformly distributed timeout values with averages of 2 (as was the setting for all previous simulation runs) and 10. We conclude from the plot that a small randomised non-zero oscillation prevention value has a small but measurable positive effect on the efficient use of counting nodes.

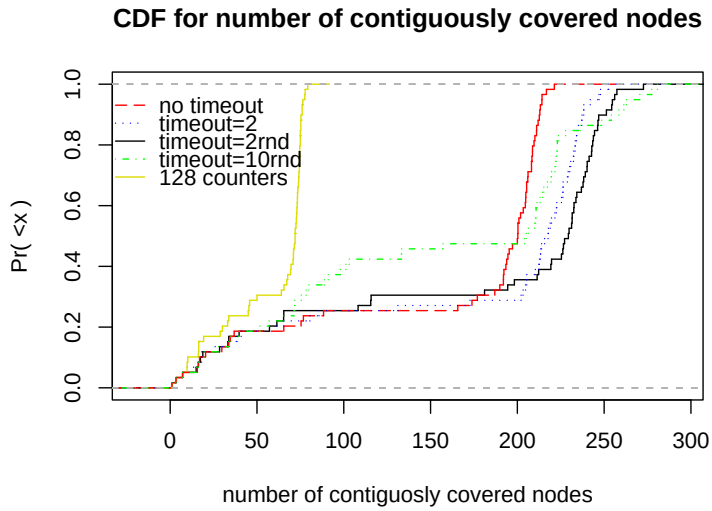


Figure 5.14: Influence of number of counters and oscillation prevention.

5.7.7 Number of counters

Fig. 5.14 shows the results for simulation runs on the default tree, but only with 128 counters available, i. e., only $\frac{1}{4}$ of those before (left line). As one can expect, the number of contiguously fully-covered nodes remains roughly linear in proportion to the number of available counters.

5.8 Conclusion

Our evaluations on simulated data confirm the theoretically derived convergence and coverage properties. Further analyses should evaluate EaC's performance on search requests resulting from real-life packet traces, applied to a search tree built from an existing real-life rule base, using a state-of-the-art packet classification method like HyperCuts [SBVW03]. Moreover, further understanding is likewise needed in how to adjust seamlessly to changes in the underlying rule-base, or how to employ effectively knowledge form previous algorithmic iterations.

As the restricted search-tree monitoring presents a negligible system overhead, it holds a significant potential for possible applications, one of them being a run-time optimisation of the search method itself. However, the gathered data can be useful in a number of ways, for example for identifying heavy flows or detecting rapid changes, failures or attacks in the network.

The presented approach can be generalised to any tree search abiding by the assumptions of downward search without backtracking. As such, this self-monitoring mechanism is a clear step towards developing a fully autonomous packet processing system that reacts dynamically to changes in the pattern of the oncoming traffic.

6 An Algorithm for Dynamic Traffic Engineering

After we have introduced methodologies for measuring and characterising traffic in the previous chapters, we now focus on the main part of this thesis: dynamic traffic engineering.

As was outlined in Section 2.3.3, current traffic engineering operates on time scales of hours. This is too slow to react to quick traffic bursts, caused by phenomena such as flash crowds or BGP reroutes. One possible solution is to use load sensitive routing. Yet, interacting routing decisions at short time scales can lead to oscillations, which has prevented load sensitive routing from being deployed since the early experiences in Arpanet [KZ89].

However, recent theoretical results have shown that re-routing policies based on game theory can provably avoid such oscillation, and that in addition can be shown to converge quickly. In this chapter we present REPLEX, a distributed dynamic traffic engineering algorithm based on this policy. Exploiting the fact that most underlying routing protocols support multiple equal-cost routes to a destination, it dynamically changes the proportion of traffic that is routed along each offered path. These proportions are carefully adapted utilising information from periodic measurements and, optionally, information exchanged between the routers about the traffic condition along the path.

6.1 Introduction

Recently, a number of promising dynamic traffic engineering or dynamic routing solutions have been proposed (e. g., TeXCP [KKDC05], MATE [EJLW01]). These systems take advantage of alternative paths in a network (e. g., MPLS multipaths) and optimise network utilisation by adjusting the distribution of traffic among the paths with the same ingress/egress nodes. These systems are not routing protocols per se; they rather operate on top of an existing routing infrastructure, which is MPLS in both cases. Accordingly, they are referred to as *traffic engineering protocols* [KKDC05], rather than routing protocols.

This chapter presents a traffic engineering protocol, REPLEX, which is applicable in a broad context: It can be deployed on top of virtually any routing infrastructure (e. g., OSPF, IS-IS, MPLS, even multipath BGP, or a network that is entirely configured by hand). The additional signalling overhead induced by our protocol is very small. Moreover, it even is not strictly necessary (although favourable) that the routers exchange in-

formation on traffic conditions; the "protocol" thus even can achieve increased network performance if the routers do not communicate with each other at all.

Our protocol is derived from an algorithm that can be proven to be stable and converge quickly in a game-theoretic model. The algorithm assumes that the only action that a router can perform is to change the traffic distribution among a set of equal-cost[1] routes between an ingress/egress pair. Similar to TeXCP or MATE, these paths are provided by an existing routing architecture.

Our algorithm specifies how a router should react to changes in its traffic load, be they external or the effect of another router's decision. In order to not reorder packets within a flow, which is known to cause bad TCP performance [LG02], traffic splitting is done on a flow-by-flow basis utilising the standard hashing technique [CWZ00]. To help the router with its decision about how to distribute the traffic among the path, the router can gather measurement data about the path itself, or, in addition, periodically exchange information with other routers. While experiments indicate that this information is helpful, the algorithm still yields improvements in cases where such communication is not desired, e. g., in an interdomain context. In general, the communication overhead is low, as we use a distance vector-like approach for distributing this information. The method is therefore scalable even for large networks.

Our proposed system features a number of novel properties. Theoretical analyses in a game-theoretic setting promise a quick convergence speed and no oscillations. A simulation-based evaluation, which we are going to present in the next chapter, supports these expectations under realistic conditions. Contrary to almost all other publications proposing dynamic routing solutions, we impose a realistic workload whose characteristics are consistent with self-similar traffic, and which is thus difficult to handle due to the resulting traffic bursts. The traffic demands are the results of heavy-tailed arrival processes of Web-like and peer-to-peer-like downloads with heavy-tailed distributions for the number of bytes and inter-download times. The Web-like and P2P-like workload moreover implies that we do neither ignore the dynamics imposed by TCP's congestion control, nor its inherent feedback to congestion in the network which interacts with any small-time scale traffic engineering. The simulations furthermore confirm that the communication overhead is low and that the system works even if no communication is possible.

One of the convenient properties of our proposed system is that it is generic with regard to the underlying routing architecture that provides the routing alternatives, may they be OSPF, IS-IS, MPLS, etc. Here, an interesting question is if the number of offered alternative paths is sufficient to enable the algorithm to do its job, or how many alternative paths are needed. Our experiments using topologies provided by Rocketfuel [SMW02] indicate that the available number of equal-cost paths when using hop-count as distance metric is sufficient.

[1] Throughout this chapter, the term *cost* refers to quasi-static costs as defined by an underlying routing protocol, e. g., OSPF link weights. It does *not* encompass variable costs that change dynamically with traffic conditions, such as, e. g., link load.

6.2 Related Work

Many works have been published on traffic engineering and related areas [FRT02, XHBN00, FT00, WXQ+06] such as traffic matrix estimation [RTZ03, ZGK+05, TDRR05, ZRLD03]. Most of the TE methods frequently used within the network operator community are offline methods: Network traffic intensities are measured over some period of time (usually a few hours); then these measurements are collected at a central point, where a new routing (e. g., readjusted OSPF weights; [FT00, FT02, BLM] and many others) is calculated. This new routing is then applied to the network, and the network remains in the new state—unless a link or a router fails, or the TE is re-run.

Such TE schemes are used by quite a number of operational networks, since they achieve good performance gains. By running TE mechanisms multiple times per week or even per day, Internet providers can react to long-term changes in traffic demands (e. g., day-of-week effects and to some extent circadian effects). Hao and Ito even propose a mechanism for dynamically adjusting OSPF weights [HI05]. However, changing weights too often increases the number of times that the intradomain routing protocol has to converge, and has an impact on the interdomain routing. During the interdomain routing convergence time, albeit in the order of seconds or milliseconds, routing loops can occur. This, in turn, leads to thousands of TCP connections suffering packet losses and falling into their slow-start phase simultaneously. Due to resulting synchronisation effects, massive service degradation can occur even well after the loops have vanished. Moreover, changing IGP weights can result in changes to the BGP routing [Tei05] and thus affect other providers and potentially be harmful to the global BGP stability.

Obviously, offline methods that are run once every few hours cannot react to sudden traffic changes in real time. Such changes can, e. g., be caused by BGP reroutes, flash crowds, malicious behaviour, and to some extent by diurnal traffic variations. Furthermore, while current TE can deal with network failures by pre-computing alternative routings, it usually does so only for a limited set of failures (e. g., explicitly via MPLS backup paths or implicitly [FT02, AC03]). It thus may fail to anticipate certain failures, which can lead to bad performance even when the network actually provides enough capacity. Even a guaranteed worst-case behaviour [WXQ+06] still leaves room for improvement.

In the past, various methods for online traffic engineering, in other words, routing protocols that react to changes in traffic demands, have been proposed. The earliest approaches to such load-adaptive routing were used in the Arpanet but showed heavy oscillations due to interactions between the decisions made by the various routers. Even though they were later revised and improved [KZ89], their tendency to oscillate remained. Due to these bad experiences, routing in the Internet was designed not to react to traffic conditions, but only to topology changes. Instead, the mechanisms to protect the network from overload were moved from the routing protocol into the end hosts.

In today's Internet, one can therefore presume an even higher danger of oscillations when deploying a load-adaptive routing protocol: A large share [DFM+06] of today's traffic consists of TCP connections, which themselves use end-to-end feedback loops in order to avoid network congestion [Pea81]. Thus a routing protocol reacting to traffic

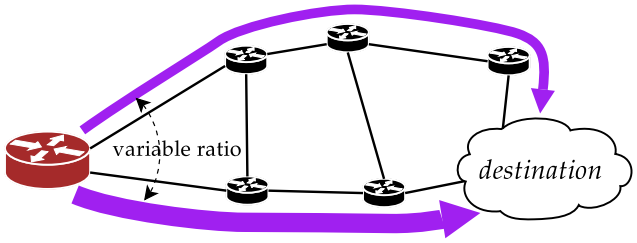


Figure 6.1: Traffic engineering using multipaths. Traffic is always sent along the same, unchanging paths; however, the share of traffic sent over each path is variable. Traffic is dynamically shifted between paths so as to increase network performance.

changes may affect a large number of TCP connections, which results in interactions between the feedback loop of the routing protocol and the feedback loops of the affected TCP connections. Hence, by reacting to changes in the traffic demands, the traffic demands themselves may change.

With all these issues in mind, network operators are sceptical about employing load-adaptive routing protocols. Instead, they use traditional offline TE techniques, combined with vast overprovisioning of their network backbone infrastructure. While this "brute-force" approach can indeed protect against potentially disrupting changes in traffic patterns, it brings about the question whether a currently overprovisioned network backbone can be used in a more efficient way, thereby allowing the operator to increase its revenues without the need for costly upgrades to its hardware.

The reluctance of network operators to employ load-adaptive routing in their network is amplified by the fact that most load-adaptive schemes that have been proposed in theory (for an overview, we refer to [Skr06]]) require them to entirely replace their current routing architecture with a new load-adaptive one. To remedy this aspect, approaches that allow automatic online traffic engineering on top of an existing traditional routing infrastructure have been proposed. Among these are MATE [EJLW01] and TeXCP [KKDC05]. Both systems represent approaches similar to ours, in that they are not routing protocols, but traffic engineering protocols [KKDC05]. They split traffic among multiple paths that an underlying routing architecture has established between the same pair of edge routers (the authors suggest MPLS tunnels), and adjust the splitting ratio dynamically depending on the traffic—see Fig. 6.1 for a graphical explanation.

MATE
TeXCP

In contrast to TeXCP and MATE, the system we propose is not restricted to path-centred (i. e., MPLS-like) scenarios. Not only is there a large number of ISPs who do not use MPLS, but also there are known issues with MPLS paths across AS boundaries [PUB04]; therefore these TE protocols are not suited as interdomain TE protocols. Moreover, our approach allows greater scalability for large networks, since the communication cost incurred at each router is linear with the number of interfaces times at most the number of destination prefixes, which can be greatly reduced by grouping prefixes that share the same set of egress nodes (Section 6.4.3). In contrast, for solutions employing virtual links such as MPLS tunnels, it is quadratic with the number of edge

routers. Finally, an evaluation of their systems' behaviour under realistic traffic load, containing TCP feedback loops and bursts caused by the self-similar nature of Internet traffic, has not been performed yet.

The Traffic Engineering extensions for OSPF routing (OSPF-TE [KKY03]) allow the routers to inform the network not only about availability and costs for each link, but also provide a framework to broadcast traffic conditions for each link. Considering that OSPF is a link-state protocol, one can expect a sizable increase in communication due to periodically broadcasting traffic conditions for each single link. The system that we propose is expected to scale better than OSPF-TE, since it aggregates information for paths in a distance-vector like fashion. Moreover, our approach does not prescribe the operator to use OSPF; rather it can even be deployed on top of routing protocols such as BGP/MIRO [XR06] across AS boundaries for the purpose of interdomain traffic engineering.

AMP

Gojmerac et al. present AMP, an interesting distributed dynamic traffic engineering algorithm which bears some resemblance with our system, since it also makes use of multipaths that an existing routing infrastructure provides [GZRR03]. They take a different approach in that their system uses so-called *backpressure messages* that recursively propagate aggregated information to upstream nodes, telling them how much their traffic contributes to congestion. Their implementation and evaluation is sound; realistic traffic and avoiding packet reordering are taken into account. However, the AMP backpressure messages do not distinguish between different traffic destinations.

DATE [HBCR07] (Distributed Adaptive Traffic Engineering) by He et al. is a new approach that explicitly takes into account the fact that the TCP traffic used in today's networks has a feedback loop whose decisions can interact with traffic engineering. They unite the network layer (traffic engineering) and transport layer (TCP congestion control) into a multi-layer approach that optimises the goals of end users and those of network operators at the same time. The system is stable even with stochastic traffic fluctuations and reacts quickly to bottlenecks, which is shown through theoretical analyses and simple simulations.

Basu et al. propose a routing algorithm where each node in the network is assigned a certain *potential*, which reflects the congestion experienced at this router [BLR03]. Their algorithm subsequently tries to route packets probabilistically, so that they avoid routers having high potentials, i. e., suffer from congestion. The drawback of this approach is the ignorance of the fact that one link at a router may be congested while another one may be not. Furthermore, their implementation does not take packet reordering into account.

The theoretical background of our algorithm is described in [FRV06, FV04, FV05]. Therein, a very similar policy is analysed in what has become known as the Wardrop model [BMW56, RT02, War52]. Ragunathan et al. analyse the performance of STARA routing protocols and propose various Wardrop-based improvements, most notably PSTARA [RK04, RK05, Skr06]. The authors focus on mobile networks where network sizes are small; realistic traffic workload and TCP issues are not considered. Liu and Reddy analyse a similar adaptive mechanism [LR07] in a multihoming scenario [LR07].

PSTARA

Adaptive routing policies have also been studied from the perspective of online

learning, where one aims at minimising the *regret* which is defined as the difference between a user's average latency over time and the latency of the best path in hindsight (see, e. g., [AK04, BEDL05]). Analyses of the convergence time for selfish load balancing problems in discrete models have been performed mainly for simple networks consisting of parallel links [EDM05, BFG+06].

In [KV05], Kelly and Voice present an analysis based on fluid models that examines the time scale on which an adaptive routing system can operate without interfering with TCP. They conclude that this is possible on the time scale of round-trip times.

6.3 Methodology

In this section we introduce our load-adaptive multipath rerouting algorithm. We start from a simple rerouting policy in a game theoretical model which can be proven to quickly converge to a stable state. This model assumes that a set of agents is located at the ingress of the network, each of which manages the path for a small amount of traffic headed towards some destination. Each agent has full control over which path its traffic takes and uses this control to minimise the latency that its traffic will sustain.

Subsequently, we adopt this algorithm from an artificial setting to an algorithm that is deployable in networks with a traditional IP routing infrastructure. We assume that path alternatives (i. e., multipaths) exist, but that the router may not necessarily have control over the path the packets will take beyond the current router (such as in OSPF equal-cost multipath). Moreover, we generalise the optimisation TE goal and also allow optimisation objectives other than the traditional minimisation of latencies.

6.3.1 Wardrop Equilibria and optimality

The algorithm we present in this section is built upon the foundation of a well-known game theoretic model, the Wardrop model. A rigorous analysis in this model proves convergence even with stale information and yields polynomial upper bounds on the time to reach approximate equilibria [FRV06, FV05]. Note that the algorithm presented in this section cannot be directly used as a routing policy in a real-world IP network. Rather, it serves as a starting point for a scheme that we will devise in Section 6.4.

Wardrop model
agent
flow
commodity
flow demand
strategy
strategy space
flow vector

In the Wardrop traffic model [War52], one assumes an infinite number of selfish *agents*, each of which wants to send an infinitesimal amount of traffic (called *flow*) through a network $G = (V, E)$. Each agent belongs to a *commodity* i from a set $[k] = \{1, \ldots, k\}$. A commodity is specified by a source s_i, a sink t_i, and a total *flow demand* d_i that is to be routed from s_i to t_i. The total flow demand d_i can be interpreted as the number of agents belonging to commodity i. We normalise the demands such that $\sum_{i \in [k]} d_i = 1$. Each agent can choose the path for their flow from a set $\mathcal{P}_i$ of paths connecting s_i and t_i. The path an agent picks is also referred to as their *strategy*, and $\mathcal{P}_i$ is therefore the *strategy space* of commodity i. By $\mathcal{P} = \cup_{i \in [k]} \mathcal{P}_i$ we denote the set of all possible routing paths. An assignment of agents, or, equivalently, traffic, to paths induces a *flow vector* $(f_P)_{P \in \mathcal{P}}$. This flow assignment in turn induces latencies on the edges.

For $e \in E$, let $f_e = \sum_{P \ni e} f_e$ denote the flow on edge e. Then the *latency* of edge e is given by the value of the respective non-decreasing *latency function* $\ell_e(f_e)$ and the latency of a path P is then $\ell_P(f) = \text{agg}_{e \in P}\{\ell_e(f_e)\}$ where agg is some *aggregation function*. In the Wardrop model, agg is typically assumed to be the sum; this is appropriate if ℓ denotes latency. If ℓ denotes any other metric like link utilisation, agg can also be the maximum or some other function. In the following, when f is clear from the context, we omit f as an argument to ℓ.

latency
latency function
aggregation function

In picking a routing path $P \in \mathcal{P}$, the agents strive to minimise the latency they sustain. From the game theoretic perspective, one is usually interested in equilibrium solutions, i. e., an assignment of agents to paths such that no agent has an incentive to deviate from the path it is assigned to. This notion is formalised by the solution concept of *Nash equilibria*, also referred to as *Wardrop equilibria* in this model.

Wardrop equilibrium

Definition 6.3.1. *A flow vector f is at a Wardrop equilibrium if for each commodity $i \in [k]$ and each path $P \in \mathcal{P}_i$ with $f_P > 0$ it holds that $\ell_P(f) \leq \ell_{P'}(f)$ for all $P' \in \mathcal{P}_i$.*

Wardrop equilibria are meaningful in a competitive scenario where each agent strives to minimise their own latency selfishly. In contrast, the operator controlling an autonomous system (AS) is usually interested in obtaining a *system optimal flow*, which minimises the total or average cost $\bar{\ell}(f) = \sum_{e \in E} f_e \ell_e(f_e)$ without any need for "competition" among its network components. Therefore, let us remark that system optimal flows are Wardrop equilibria with respect to modified latency functions. More precisely, a flow at Wardrop equilibrium for so-called *marginal cost* [BMW56] latency functions $h_e(x) = (x \cdot \ell_e(x))\frac{d}{dx}$ is known to be system optimal for the original latency functions ℓ_e. Note that for linear latency functions without offset, i. e., $\ell_e(x) = a_e \cdot x$, we have $h_e(x) = 2a_e \cdot x = 2\ell_e(x)$ implying that Wardrop equilibrium and optimum coincide.

6.3.2 The Exploration-Replication policy

We are considering the Wardrop model in a dynamic, round-based variant. Here, agents are activated every T seconds and are then allowed to change their path simultaneously. An interesting question is then whether agents can distributedly and jointly learn a Wardrop equilibrium quickly. The main problem is to avoid oscillation. To illustrate this, consider the natural policy where agents migrate to a path with minimal latency whenever they are activated. Such policies are generally referred to as *best response* policies. However, best response policies lead to greatly increased congestion on the optimal path at the end of a round and cause oscillation effects subsequently.

Hence, our rerouting policy must be more careful. The so-called (α, β)*-exploration-replication policy* presented in [FRV06] is designed to avoid oscillation and finds approximate equilibria quickly. This policy is simple, easy to implement, and requires only local knowledge.

exploration/replication

Informally, our policy as performed by all agents in parallel, can be described as follows: At regular intervals an agent samples another path applying one out of two *sampling* techniques: For *uniform sampling*, which is executed with only a small prob-

sampling
uniform sampling

ability, every path is sampled with uniform probability. This sampling step guarantees that every path has strictly positive sampling probability and is consequently responsible for an *exploration* of the strategy space. For the second sampling technique, *proportional sampling*, the probability of sampling a path is proportional to the fraction of agents already using it, i. e., the popularity of a path is taken as an indicator of its quality. Thus, agents using good paths are more likely to be imitated. Proportional sampling is therefore responsible for *replication* of paths with small latency and will thus "boost" successful strategies. It can indeed cause the flow on such a path to grow exponentially fast.

exploration

proportional sampling

replication

Having sampled a path Q, the agent must eventually decide whether or not they want to migrate from their old path P to Q in a *migration* step. Again, the decision is randomised. Indeed, the migration probability is sensitive to the relative latency gain that can be achieved by switching from P to Q. More precisely, the migration probability is

migration

$$p_{\text{migration}} = \max\left\{0, \frac{\ell_P - \ell_Q}{\ell_P + \alpha}\right\} \quad .$$

Here, α is some parameter that can be interpreted as a positive offset added to all latency functions to prevent the migration probability to become overly large if ℓ_P (and also ℓ_Q) are close to zero.

More formally, our policy can be described in the following way. In every round, every agent is activated with probability λ. Each activated agent performs the following two steps (consider an agent in commodity i, currently assigned to path $P \in \mathcal{P}_i$):

1. *Sampling:* With probability β, perform step (a); with probability $1 - \beta$, perform step (b).
   a) *Uniform Sampling:* Pick a path $Q \in \mathcal{P}_i$ with probability $1/|\mathcal{P}_i|$.
   b) *Proportional Sampling:* Pick a path $Q \in \mathcal{P}_i$ with probability f_Q/d_i.
2. *Migration:* If $\ell_Q(f) < \ell_P(f)$, migrate from P to Q with probability $p_{\text{migration}} \frac{\ell_P - \ell_Q}{\ell_P + \alpha}$.

The adaptive migration rule ensures that small latency gains only cause a small migration rate. This is necessary to avoid oscillation. However, the migration rate also depends on the choice of the parameter λ which consequently determines whether or not oscillation can occur. Its value has to respect the steepness of the latency functions. This is made precise by the following theorem which is (in a slightly stronger version) proved in [FRV06].

Theorem 6.3.1. *If the latency functions are polynomials of degree d and $\lambda \leq c/d$ for a suitable constant $c > 0$, the (α, β)-exploration-replication policy converges towards a Wardrop equilibrium if*

convergence

$$\beta \leq \frac{\min_{P \in \mathcal{P}} \ell_P(0) + \alpha}{\textit{maximum path length} \cdot \max_{e \in E} \max_{x \in [0,\beta]} \ell_e(x)} \quad .$$

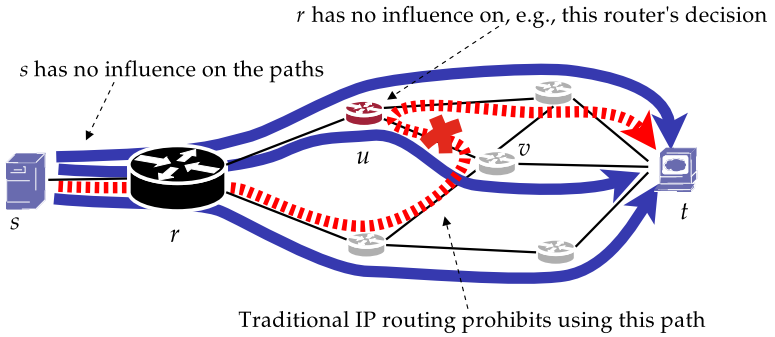


Figure 6.2: Challenges when adopting the Wardrop model to IP routing.

This theorem can actually be generalised to latency functions with finite first derivative. While this ensures convergence in the long run, in order to give a bound on the time of convergence we need to define *approximate equilibria*. Let $\bar{\ell}(f)$ denote the average latency of all paths and let $\bar{\ell}_i(f)$ denote the average latency of paths in commodity i. We say that a flow is at a *(δ,ϵ)-equilibrium* iff for every commodity $i \in [k]$, ϵ_i agents use paths with latency at least $\bar{\ell}_i(f) + \delta \cdot \bar{\ell}(f)$ and $\sum_{i\in[k]} \epsilon_i \leq \epsilon$. It can be shown that for suitable choices of parameters, the (α, β)-exploration-replication policy reaches a (δ,ϵ)-equilibrium in time

approximate equilibria

(δ,ϵ)-equilibrium

$$\mathcal{O}\left(\frac{d}{\epsilon^2\delta^2}\log\frac{d\max_f \bar{\ell}(f)}{\min_f \bar{\ell}(f)}\right) ,$$

where d is again (a generalisation of) the degree of the polynomial latency functions [FRV06]. We believe that these positive results are a strong indication that a load-adaptive rerouting algorithm based on this selfish rerouting policy can perform well in practice. In the following section, we develop such an algorithm.

6.4 Wardrop routing in an IP network

The Wardrop model draws on a rather theoretical setup: It is suitable when the number of agents is infinite, when the agents can fully control their traffic and the chosen path, and when they have easy access to the current latency of the paths. However, none of these requirements is true in real-world networks such as the Internet. In the following we address all of the aforementioned issues and show that a variant of our Wardrop policy still can yield a practical algorithm that can be deployed in practice.

6.4.1 Delegation of Decisions

The first major problem is that agents normally do not have control over the entire path their packets take in real-world communication networks—usually, the end points of a communication stream are located at the periphery of the network, and are typically

connected to the network by only a single link, e. g., Ethernet, DSL, or 802.11. Rather, it is the routers in between the endpoints that are responsible for choosing the path along which the data packets are sent. In the scenario depicted in Fig. 6.2, the Wardrop agent should naturally be associated with the data source s; however, s has no influence on the paths of the flows $s \rightsquigarrow t$ that are actually being used for reaching destination t.

Moreover, in most cases not even the routers can choose a path as a whole: Apart from cases where tunnels or virtual links such as ATM circuits or MPLS LSPs are being used, a router can only determine the next-hop node on the way to the destination if traditional IP routing is used. In Fig. 6.2, we see that, even if the Wardrop agent is located at r and not at s, it still has no influence on the forwarding decisions met at u—thus, it does not have full control on the paths to be used.

Furthermore, in normal IP routing this decision is solely based on the destination of a packet, whereas other header attributes (e. g., the source) are neglected. This has the effect that not all possible paths through the network actually can be used—the dotted line in Fig. 6.2 shows an example for a path that is not allowed in traditional IP routing protocols that are based on hop counts or link costs.

As we see, the mapping of Wardrop agents to a network is non-obvious. In this section we describe how agents (end hosts) delegate rerouting decisions to adjacent routers. These routers make partial decisions and, in turn, delegate further decisions they are unable to carry out themselves along the path.

Consider a router r that is located somewhere on a path $s \rightsquigarrow r \rightsquigarrow t$ from s to t. There may exist several paths from r to t. Applying our Wardrop rerouting policy, router r aims at distributing the traffic from s to t evenly among these paths. However, r does not know about all of these paths, but merely knows a set of possible next hops of routes for destination t denoted by $N(r,t)$. In practice, these routes can be manually configured, or they are obtained from an underlying routing protocol such as OSPF as equal-cost routes.

In order to control the traffic balance, the router r maintains a set of dynamically changeable weights $w(r,t,v_i)$ for every target t and possible next-hop neighbour $v_i \in N(r,t)$. We normalise the weights for every target t such that they all sum up to 1, i. e., $\sum_{v_i \in N(r,t)} w(r,t,v_i) = 1$. For the time being, let us interpret $w(r,t,v_i)$ as the exact fraction of traffic routed from r to t via v_i.

6.4.2 Distributing information

From the perspective of an agent at source s, the routing decision made at intermediate node r influences only the performance of the path section from r to t, whereas the performance of the section from s to r is fixed. In order to make the routing decision, r must gather traffic information about the set of paths between r and t.

On the other hand, r cannot control which one of the possible paths a packet will take once it has been forwarded to one of neighbours v_i. Thus, r cannot exploit exhaustive information about *all* possible paths. Consequently, we use only aggregated information about the possible paths from next-hop v_i to destination t. We now describe how

router r can gather this information simply by performing measurements of the adjacent links (r, v_i) and exchanging messages with its peer routers v_i. This information exchange resembles a distance vector routing protocol like RIP (Section 2.2.3 on page 23).

Since r cannot influence a packet's path beyond the chosen link to neighbour v_i, the decision whether to send the packet via link $r \to v_i$ is based on the *expected latency* $L(r, t, v_i)$ for the path $r \to v_i \rightsquigarrow t$. Router v_i can keep r informed about the expected latency of its paths $v_i \rightsquigarrow t$ by periodically sending messages. But how can r use these values to compute $L(\cdot)$? Let us consider the case that agg is the sum, and let $\mathcal{P}(x, y)$ denote the set of paths between node x and y. Then r's valuation $L(r, t, v_i)$ of the next-hop node v_i for destination t is

expected latency

$$L(r, t, v_i) := \ell_{rv_i} + \sum_{P \in \mathcal{P}(v_i, t)} w_P \cdot \ell_P$$

where w_P is the weight of path P. This means that for $P \in \mathcal{P}(u_1, u_k)$, we have $w_P = \prod_{i=1}^{k-1} w(u_i, t, u_{i+1})$. The measurement value of the next-hop edge ℓ_{rv_i} can be evaluated at r.

Let us now specify the information that router v_i sends to r. We define

$$A(v_i, t) := \sum_{P \in \mathcal{P}(v_i, t)} w_P \cdot \ell_P \ .$$

to be the average latency that we expect to obtain when r forwards traffic with destination t via next hop v_i.

We then obtain

$$\begin{aligned} A(v_i, t) &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \sum_{P \in \mathcal{P}(u_j, t)} w_P \cdot \left(\ell_{v_i u_j} + \ell_P \right) \\ &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \left(\ell_{v_i u_j} + \sum_{P \in \mathcal{P}(u_j, t)} w_P \cdot \ell_P \right) \\ &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \cdot L(v_i, t, u_j) \ , \end{aligned}$$

where the first equality is the definition of ℓ_P, the second holds because $\ell_{v_i u_j}$ does not depend on P and the weights sum up to 1, and the third equality holds as it uses the definition of $L(\cdot, \cdot, \cdot)$.

To summarise, the value $A(v_i, t)$ is computed at v_i and sent to r at regular intervals, after which r can update $L(r, t, v_i) = \ell_{rv_i} + A(v_i, t)$. $A(v_i, t)$ can be interpreted as condensed information about the performance of subsequent path sections. Note that we are using the fact that $L(v_i, t, u_j)$ is already defined for all $v_i \in N(r, t)$ and $u_j \in N(v_i, t)$ when computing $L(r, t, v_i)$. This corresponds to the information being propagated backwards along a path.

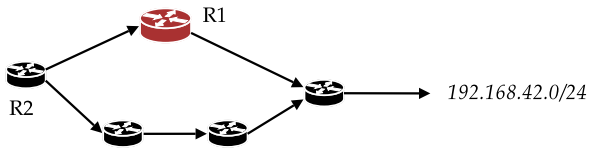


Figure 6.3: No communication restriction allowed: Even though router R1 has only one route to reach the destination prefix, it is part of a multipath from R2 to this destination. Hence, R2 is dependent on traffic information from R1 regarding this destination.

Based on the values stored in $L(r, t, \cdot)$, router r can adapt its weights $w(r, t, \cdot)$ over time and distribute the traffic of this demand (from various sources to the same sink t) as evenly close to the weights as possible. Section 6.4.4 will explain how this split can performed in practice.

6.4.3 Communication costs

Obviously, the routers need to communicate with each other if they are to base their decisions not only on local observations, but on the general state of the network. This brings about the question how much additional traffic is exchanged in course.

For every destination t, the size of such a message contains the identifier of t, which is typically an IPv4 prefix of $32 + 5$ bits, plus the value of $A(v_i, t)$ itself. If we allow 11 bits for this value, then the total size of an update message for one destination prefix amounts to 6 bytes. For IPv6 prefixes, this value roughly triples to $128 + 7 + 11$ bits ≤ 19 bytes per update message. Even if one assumes an extreme "worst-case" scenario where information about 200,000 IPv6 prefixes is exchanged every second, the resulting communication overhead is only 3.6 MBytes/s on each link. Compared to today's link capacities of 10 Gbit/s and more, this value is small ($< 0.4\%$)—even under these extreme assumptions.

Furthermore, the information exchanged between routers can be significantly reduced by a number of methods. First, no information exchange is needed for those destinations for which the underlying routing protocol does not offer any (best-cost) multipaths. Second, routers that are not part of a best-cost multipath do not require to exchange traffic information. Note, however, that routers which do not have multipath alternatives to choose from, but which are part of a multipath, still need to forward traffic information; see Fig. 6.3 for an explanation. Third, destinations that have exactly the same set of egress points from a REPLEX-controlled network share the same traffic conditions along their paths. Assuming normal IP routing, the paths to these egress routers always remain the same, regardless of the destination behind the router.[2] Therefore, traffic information only needs to be sent once for each of these egress set equivalence classes; see Fig. 6.4 on the next page for an explanation.

To conclude, REPLEX incurs only minimal additional communication cost.

[2]Of course, the conditions on their further path outside the REPLEX-controlled network are likely to differ, but we assume that we cannot obtain traffic information from outside our REPLEX-enabled network.

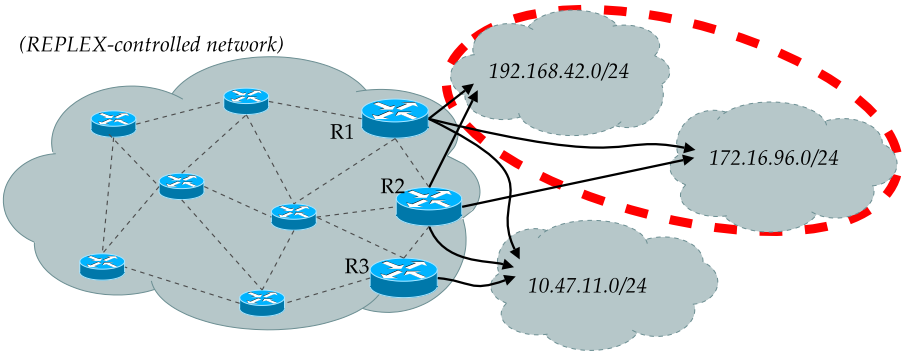


Figure 6.4: Saving communication costs: The `192...` and the `172...` destinations have exactly the same egress points, i. e., R1 and R2; hence their REPLEX traffic information can be aggregated. In contrast, the `10...` destination has R3 as an additional egress point; thus its traffic information cannot be aggregated together with the other two.

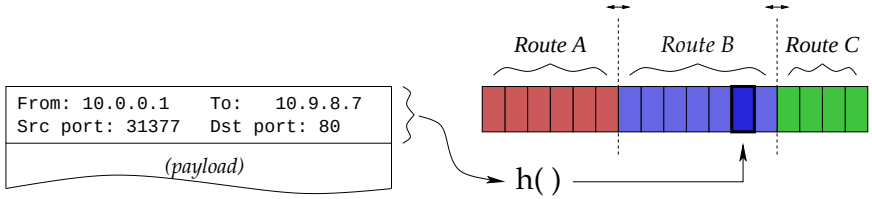


Figure 6.5: Avoiding packet reordering using hashing. Traffic is shifted between routes by adjusting the boundaries between the bins for the hash values.

6.4.4 Randomising vs. hashing—ruling out packet reordering

A natural way of distributing traffic according to some weights $w(r, v_i, t)$ is to use simple randomisation, where each weight is interpreted as a probability. However, this probabilistic routing approach has the drawback that packets headed towards the same destination may take different paths—a bad decision. Even though the *expected* latencies along each path may be the same, the *actual* latencies can differ. Thus packets can overtake each other, which results in packet reordering at the destination. Since TCP treats packet reordering in the same manner as it treats packet losses, probabilistic routing can seriously harm TCP performance [LG02].

Therefore we "roll the dice" not per-packet but rather per-flow by applying the well-known technique [CWZ00] of using a hash function $h : V \times V \mapsto [0, 1]$ mapping each packet based on its source and destination address to some value. For every destination t, we partition the interval $[0, 1]$ into sections of size $w(r, t, v_i)$ and label each interval with the corresponding next-hop node v_i. Whenever a packet from source id_s and destination id_t arrives, it is forwarded to the node associated with the interval containing $h(id_s, id_t)$—see Fig. 6.5.

hash function

As long as the weights are constant, we can thus be sure that no packet reordering occurs. Whenever weights are shifted, this causes a fraction of the traffic to be rerouted, *possibly* causing packet reordering. Hence, the time interval at which weight shifts occur

should not be smaller than the time it takes for a typical TCP connection to recover from packet losses, i. e., on the order of several hundred milliseconds.

Let us remark that if there is a range of hash values that are systematically more popular than others, be it by accident or due to an attempted denial-of-service attack against our REPLEX network, this causes our weight shifting procedure to decrease the corresponding weight accordingly.

Note that popular hash methods as they are implemented in practice typically do not have the continuous interval $[0, 1]$ as their co-domain, but rather some (small) subset $H \subsetneq \mathbb{Z}$. Our calculated weights $\in [0, 1]$ thus need to be scaled and discretised in some form, such that their sum amounts to $|H|$. The similar problem occurring in democratic elections, i. e., determining the number of seats for each party based on a much larger number of votes, is resolved by *apportionment* methods. Several apportionment methods are known, e. g., the largest remainder (Hare-Niemeyer or Hamilton's method), d'Hondt (Jefferson's method), Sainte Laguë/Schepers, Hill-Huntington (U.S. congressional appointment), and some more (for an overview, we refer the reader to [Bog, Wik07a, Wik07b]). Luckily, democratic fairness aspects such as malapportionment, monotonicity, consistency etc. only have marginal effects on the assigned hash values; thus we can safely ignore them in our case. Therefore we simply decided to use the method that incurs the lowest CPU overhead. This method proved to be the largest remainder method.

apportionment

6.4.5 Measuring performance

In Section 6.4.1 we have assumed that every router r is able to measure ℓ for all outgoing edges $\{r, v_i\}$. With an acceptable computational effort we can measure the following values during each time interval: the number of packets sent and dropped (due to link overload), the number of bytes sent and dropped, and the average queue length. From this information we can, e. g., compute the following metrics:

latency

1. The *latency* as the sum of the (constant) propagation delay of the edge plus the (variable) queueing delay. Obviously, latency values are additive along a path.

link utilisation

2. The *link utilisation*. Let b denote the bit rate of an interface, T the length of the measurement interval, and let n denote the number of bytes sent within this interval. Then we compute the link utilisation as $\frac{8 \cdot n}{T \cdot b}$. We define the utilisation of a path to be the maximum utilisation of a link in the path, i. e., the aggregation function agg is the maximum.

3. The *packet loss probability*, short *loss rate*. We compute this probability p as the number of dropped packets per interface, divided by the number of packets that were meant to be sent out via that interface. In order to obtain the packet loss probability along a path, we can calculate its opposite, i. e., the packet delivery probability, $(1 - p)$. We then can define the packet delivery probability along the path as $1 - p_{path} = \prod_{router \in path}(1 - p_{router})$ and thus $p_{path} = 1 - \prod_{router \in path}(1 -$

p_{router}). In other words, the aggregation function agg is the product of the additive inverse in this case.

Different applications require optimisation of different parameters. Applications like voice over IP require small latency, accessing files requires large throughput, other applications may require high reliability (i. e., small packet loss probability), and a typical goal of an ISP doing traffic engineering is to minimise the maximum utilisation of an edge in the network. In this work, we focus on the latter. Let us remark that in this case, Wardrop equilibria with respect to link utilisation are also optimal from the ISP's point of view. Note, however, that our algorithm can cater for other metrics as well.

Internet traffic is known to be very bursty [PW00]. Therefore, traffic measurements made within a short time interval are not very reliable. To this end, instead of using the measured values directly, we use an exponential moving average (EMA). Let $\tilde{\ell}(r,v)$ be our current metric value for link (r,v), and suppose that we measure a value of ℓ_{rv}. Then we update the value of $\tilde{\ell}(r,v)$ in the next round to be $\tilde{\ell}(r,v) \leftarrow \eta \cdot \ell_{rv} + (1-\eta) \cdot \tilde{\ell}(r,v)$.

6.5 The REPLEX Algorithm

Combining the techniques presented in the previous section, we now present the protocol which is run on each individual router that participates in a network optimised by our algorithm. Since it is derived from the exploration-replication policy, we name our algorithm REPLEX.[3] Recall that a protocol implementation has to address several tasks: Foremost, it needs to calculate the weights $w(\cdot)$. For this, measurements $\ell_\cdot$ on the router's link queues are necessary. These are combined with the information $A(\cdot)$ that is received from neighbouring routers. Finally, it has to compute the information $L(\cdot)$ which it then sends to its neighbours. In order to perform above computations, each router r maintains the following arrays:

REPLEX

$\tilde{\ell}(r,v)$**:** The EMA of the measurements for link (r,v).

$L(r,t,v)$**:** metric value for destination t using next hop node v

$A(v,t)$**:** The average measurement value that next hop router v has announced for destination t. This array is updated whenever update messages from neighbouring routers are received. It is initialised with values that are neutral w. r. t. $\mathrm{agg}(\cdot,\cdot)$.

$w(r,t,v)$**:** Current weight of route $r \to v \rightsquigarrow t$ (i. e., route for destination t via next hop neighbour v)

In the router's main loop, which is executed every T seconds, these values are updated and sent to the neighbouring routers, as described in Section 6.4.1. Algorithm 4 describes the procedure in more detail.

At the heart of the algorithm we have the actual weight adaption procedure described in Algorithm 5. For every pair of next hop nodes, this procedure computes the

[3] ... which happens to sound better than EXREP.

Algorithm 4 Main loop of the REPLEX algorithm.

1: initialise an empty message M
2: **for** each destination t in the routing table **do**
3: **for** all next-hop nodes $v \in N(r,t)$ **do**
4: measure performance ℓ_{rv}
5: $\tilde{\ell}(r,v) \leftarrow \eta \cdot \ell_{rv} + (1-\eta)\tilde{\ell}(r,v)$ // *calculate EMA*
6: $L(r,t,v) \leftarrow \text{agg}(\ell_{rv}, A(v,t))$.
7: **end for**
8: $avg \leftarrow \sum_{v \in N(r,t)} w(r,t,v) \cdot L(r,t,v)$
9: Append (t, avg) to M
10: **end for**
11: send M to all neighbours v who store it in $A(r,\cdot)$
12: call procedure ADAPTWEIGHTS (Algorithm 5)

Algorithm 5 Procedure ADAPTWEIGHTS.

1: **for** each destination t in the routing table **do**
2: $w'(r,t) \leftarrow w(r,t)$.
3: **for** each pair of next-hop routers $v_1, v_2 \in N(r,t)$ **do**
4: **if** $L(r,t,v_1) > L(r,t,v_2) + \epsilon$ **then**
5: $\delta \leftarrow \lambda \left((1-\beta) w(r,t,i) + \frac{\beta}{|N(r,t)|} \right) \frac{L(r,t,v_1) - L(r,t,v_2)}{L(r,t,v_1) + \alpha}$
6: $w'(r,t,v_1) \leftarrow w'(r,t,v_1) - \delta$
7: $w'(r,t,v_2) \leftarrow w'(r,t,v_2) + \delta$
8: **end if**
9: **end for**
10: set $w(r,t) \leftarrow w'(r,t)$.
11: **end for**

migration rates (line 5) similar to the policy described in Section 6.3.1. The computed rates are then migrated by shifting the corresponding weights.

Parameters

Obviously, the REPLEX algorithm depends on a number of different parameters, which have been introduced to in the previous subsections. In the following we give a concluding overview on the static parameters of our algorithm.

update period length T: The length of the interval at which the main loop is executed.

communication period length T_{comm}: If communication between the routers is decoupled from the main loop (not shown in algorithm 4 for the sake of simplicity), this value describes the time intervals between subsequent messages to neighbouring REPLEX instances.

weight shift factor λ: This parameter determines the weight shifted in one round. The ratio λ/T controls the convergence speed of the algorithm. Oscillation effects and congestion control feedback loops of affected TCP connections limit the maximum convergence speed that we can achieve.

EMA weight η: Decreasing the weight of the exponential moving average makes the algorithm less sensitive to the effects of bursty traffic, whereas choosing it too small increases the time span until the algorithm realises the effects of rerouting decisions, or reacts too slowly to traffic changes.

virtual latency offset α: This virtual offset is added to all path latencies $L(r,t,v)$ making the algorithm less sensitive to small differences when the metric is close to 0. (For a detailed discussion, see [FRV06].)

improvement threshold ϵ: The optional parameter ϵ can be considered to be a damping factor. Projected weight shifts that are smaller than ϵ are not carried out; thus one can ensure that weights are only shifted if the change is substantial.

exploration ratio β: This parameter determines the ratio between replication and exploration. If β is too small, currently unused paths may not be sampled by our algorithm. On the other hand, β should not be too large, since this can result in excessive growth of a flow if f_P is close to zero. (For a detailed discussion, see [FRV06].)

metric to optimise: As we pointed out before, REPLEX is not restricted to operate with latencies. Rather, the REPLEX instances in a network also can be used to optimise other network performance metrics, e. g., maximising throughput, minimising link loads, or minimising packet losses.

nonlinear link load parameter κ: This parameter is specific to the nonlinear link load and will be introduced in Section 7.1.6.

Our simulations show that the performance is largely independent of the choice of α, β and ϵ, as long as these are chosen within reasonable intervals. We find $\alpha, \beta \in [0.1 \ldots 0.25]$ and $\epsilon \in [0 \ldots 0.05]$ to offer a good trade-off between the effects described above. The more sensitive parameters that influence the performance most are T, λ, η, and the metric that the individual REPLEX instances aim to optimise. Accordingly, we explore this parameter space in the following chapter in greater detail.

7 Evaluation

In this chapter, we evaluate the REPLEX algorithm that was introduced in the previous chapter via network simulations. It is our intention to show that our protocol behaves well with realistic traffic. Therefore, we employ traffic loads that mimic actual Web traffic, i. e., bursty TCP traffic whose characteristics are consistent with self-similarity and with those shown in Chapters 3 and 4.

As predicted by theory, the simulations quickly converge and do not exhibit significant oscillations. This holds both for artificial as well as real topologies.

7.1 Simulation setup

The purpose of our simulations are twofold: On the one hand, we use them to determine good parameters for the algorithm. On the other hand, we want to understand the behaviour of the system both in simple scenarios as well as in more complex ones. Accordingly, we consider various topologies ranging from simple four-node graphs to complex intra-autonomous system topologies such as those provided by the *Rocketfuel* project [SMW02]. The specifics of the topologies are discussed when we present the results. However, the principal workload and routing setup is the same across all simulations, and is summarised in the following subsections.

Rocketfuel

7.1.1 Network simulator

For our simulations we use the toolkit SSFNet (version 2.0, Java) [Ren], since it encompasses the features that we need: a full TCP implementation, flexible workload generators, complex topologies, and support of multiple routing protocols (including OSPF [BÖ3], MPLS [Kra03], and BGP [Ren]). Furthermore, it allows us to run complex simulations lasting for multiple hours of simulated time with ten to one hundred routers and several hundreds of clients within reasonable computation time (hours to a few days), and with thousands of clients and hundreds of routers in not quite so reasonable computation time (many days to weeks).

Within each simulation run, we distinguish three phases: a *startup phase* which in most cases lasts 1,000–30,000 seconds, a *comparison phase* of 4,000 to 10,000 seconds where REPLEX is not enabled yet, and an *evaluation phase* of another 4,000 to 10,000 seconds during which we study the performance of the algorithm. The startup phase is used to setup the underlying routing system and to enable the workload generator to reach a "stable" state, while the comparison phase allows us to compare the performance of the network before and after (i. e., evaluation phase) the start of REPLEX.

startup phase
comparison phase
evaluation phase

7.1.2 Routing

As our algorithm is not a routing protocol but a traffic engineering protocol, it relies on an underlying routing system. We use the SSFNet OSPFv2 implementation [BÖ3] for this purpose. The computation of the routes is performed right at the start of the simulation and only takes a few seconds.

Some of the Rocketfuel maps are shipped with additional Rocketfuel-deduced IGP weights. Note that these weights are presumably the result of some traffic engineering process. As we do not know the traffic matrix underlying this TE process, it does not make sense for us to use these IGP weights in our simulations. Instead we use uniform OSPF weights, resulting in hop count as the distance metric. Using hop count increases the number of equal-cost paths when compared to routing based on, e. g., weights inferred from Rocketfuel measurements. Therefore it provides the algorithm with more flexibility.

7.1.3 Realistic workload

Overall, it is well known that traffic volume [WPT98] and flow arrival streams [Fel00b] are self-similar and exhibit significant burstiness. To account for this bursty nature of traffic on the Internet [PW00] and the fact that most traffic on the Internet is still using TCP [DFM+06], we choose to generate our workload using the Web workload generator of [Vol04]. This generator uses a workload model that is similar to that of the Web workload generator that is shipped with SSFNet [Ren], but offers additional features. It uses a workload model that is similar to the one introduced by SURGE [BC98] or NSWeb [Wal01]. We parametrise the workload generator with data from [Bar01], in order to generate TCP traffic with realistic properties [FGHW99]. A survey conducted by Schneider [Sch06] confirmed that these parameters still can be considered to be consistent with today's Web traffic. To further increase burstiness, we purposely disable the use of persistent HTTP connections [KR01].

Our generator simulates Web users that read a Web page, then are idle for some time before reading another Web page, become idle again, etc. Therefore, one has to allow the system to reach its equilibrium during a certain startup phase. We achieve this by starting all clients at uniformly distributed random times (to avoid synchronisation effects) after the route computations have finished. In most cases, the clients are started in the simulation time interval between 50 s and 10,000 s. After all clients have been started, and before the startup period ends, we wait for another 2000 s so that the traffic can stabilise, before we end the startup phase and continue with the actual evaluation phase. The different phases of the simulation are shown in Fig. 7.1.

Like the timeouts, the simulated file sizes are also heavy-tailed. Thus, they yield many short flows, as well as a significant number of long-lasting flows.

In today's Internet though, the Web is not the main traffic contributor any more: In the last couple of years, it has been de-throned by peer-to-peer filesharing networks, such as Gnutella [SZR06], BitTorrent [IUKB+04], eDonkey [Tut04], Kademlia [MM02], etc. All in all, filesharing networks contribute more than 50 % of today's Internet traf-

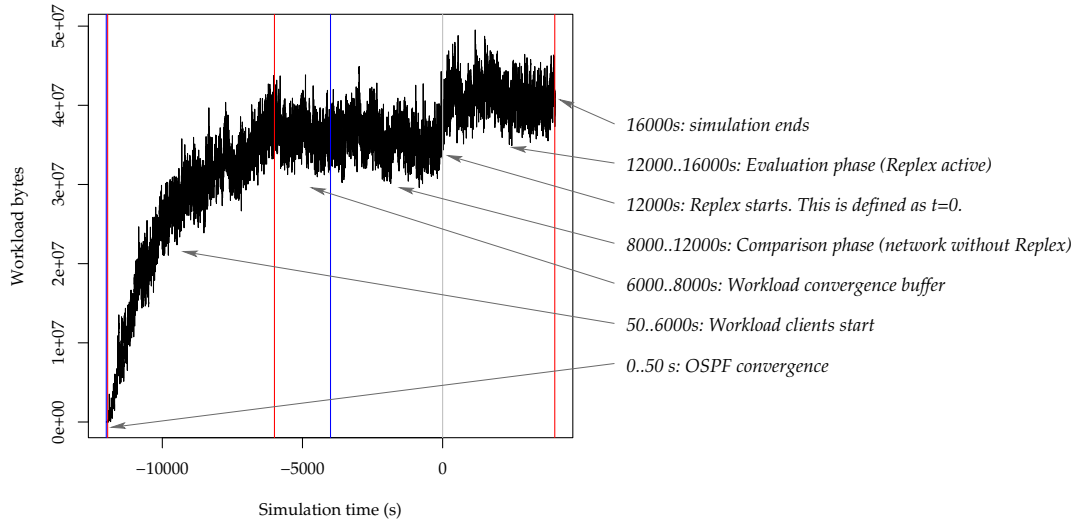


Figure 7.1: The different phases in simulation time and their influence on the number of bytes transmitted through the network in an example simulation. Note that we define $t = 0$ in simulation time to be the start of REPLEX, in order to ease comparisons across simulations that use different startup period lengths. The numbers on the right, in contrast, show the absolute simulation time counted from the start of the simulation.

fic [KBB+04]. Moreover, their characteristics differ significantly from those of Web traffic [SZR06, LBBSS02].

We conjecture that it is sufficient for our purpose of generating workload to simulate only the actual downloads, while leaving out additional traffic such as searches, since the major part of peer-to-peer traffic is related to downloads. Moreover, most of them use HTTP as a transport protocol. Therefore, although simulators for peer-to-peer networks exist [AFM05], we simply add a second group of "Web" clients and servers that we use to simulate peer-to-peer file transfers. This peer-to-peer group of "Web" clients and servers is independent of those Web traffic hosts that are responsible for the *actual* Web traffic, and we change the model parameters for the new group to imitate typical peer-to-peer network characteristics [SZR06, SR06] instead of Web traffic. Foremost, the larger file sizes of filesharing traffic [LBBSS02, AdMD+04] lead to much longer-lasting TCP connections when compared to those of Web traffic. Note that the these longer-lasting TCP connections will typically spend most of their time in the congestion avoidance phase instead of the slow-start phase [Pea81]. Therefore, they potentially suffer even more from eventual packet reordering caused by REPLEX path weight changes than the shorter-lasting Web TCP connections.

We note that alternative approaches—such as replaying a pre-recorded trace, or generating packets with randomised inter-packet timeouts following some heavy-tailed distribution—do not suffice, as they do not account for interacting feedback loops.

After all, the self-adjusting traffic engineering protocol *as well as* other control loops, foremost TCP, *both* react to congestion. Therefore, it is crucial to investigate potential interactions of the TCP congestion control feedback loop with the actions of our TE protocol: While it has been shown that a joint system of (traditional) traffic engineering and TCP congestion control can be stable under specific conditions, this is not necessarily the case with REPLEX as well. Rather, the doubled feedback could easily lead to overreactions, which in turn can cause oscillation—even when REPLEX's decision strategy just by itself would not cause any harm.

Obviously, simulating TCP plays an important role in the generation of our realistic traffic workload. However, many different TCP implementations exist in the real world [PAD+99, YQL04, Dir] and, moreover, many of these offer configuration options that affect their behaviour (e. g., enabling or disabling the Nagle algorithm [Nag84]). Since we are interested in identifying potential performance losses that might arise through adverse interactions of REPLEX and TCP, we decide to tune TCP towards a greater sensitivity to packet reordering. Although our hashing mechanism ensures that all packets pertaining to one TCP connection are sent along the same path, it may happen that REPLEX meets a decision to assign a subspace of the hash values—and thus a number of currently active TCP connections—to a different route. In an unlucky case featuring extreme differences in path latency, this can result in one pair of packets per TCP connection being shuffled.

The TCP implementation of SSFNet simulates an older BSD-style TCP implementation [Ren]; hence reordered packets are regarded as packet losses. We thus disable the Fast Recovery mechanism in TCP [MA99] on purpose, in order to increase the adverse effect of packet losses, as well as of packet reordering, on TCP throughput.

In order to add variability to the experienced RTT values in the network [FGHW99], we randomise the link delay for each client and each server. They are chosen uniformly from the interval $[0, 60]$ ms for clients and $[0, 25]$ ms for servers. Also for the sake of variability, we connect clients and servers with random bandwidths to their router. The random bandwidths are chosen uniformly from $[0.1, \ldots, 2]$ times the default link bandwidth of the scenario (i. e., 1 Mbit/s to 20 Mbit/s in most of our scenarios).

All in all, our Web workload generator produces realistic bursty workload traffic. For illustration, Fig. 7.2 shows the average load for a simulated 10 Mbit/s link that is traversed by the traffic from 100 clients and five servers, with time resolutions ranging from fracetions of a second to several minutes. Obviously, the traffic on the link fluctuates heavily, which is both due to the self-similar characteristic of the random distributions employed in the workload model, as well as to the competing feedback loops of concurrent TCP connections that interact with each other.

7.1.4 Network topology

We evaluate REPLEX both in simple artificial, as well as in more complex realistic topologies. Further details on the simple topologies are given below, together with a discussion of the simulation results.

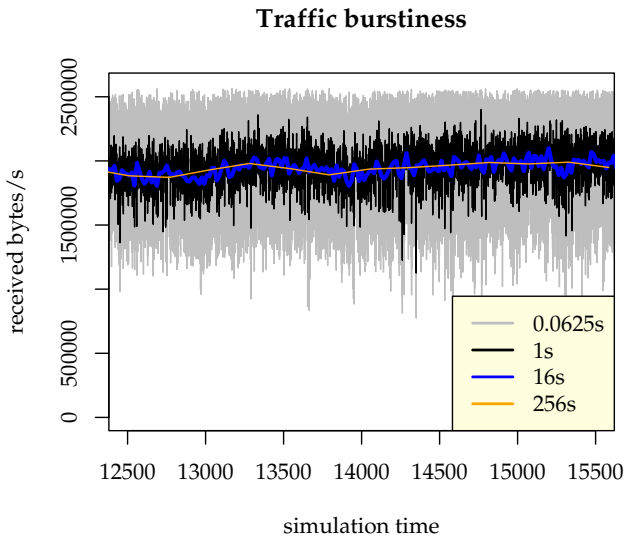


Figure 7.2: Example for bursty workload behaviour. The traffic on the link fluctuates heavily. Fluctuations are even visible on long-range timescales, such as 16 and 256 seconds. 1 simulation run, REPLEX not active.

Our realistic topologies come from two sources. Our main source is the Rocketfuel project [AMSW, SMW02]; the other is the AT&T topology that is shipped with the Totem traffic engineering software toolbox [BLM].

7.1.5 Limitations of simulation scale reduction

Obviously, when compared to real-world networks, our simulated networks are significantly smaller in size: It is just not feasible to simulate the entire Internet or even a large fraction of it, involving millions of hosts and each generating realistic TCP traffic. We therefore have to take care that we do not introduce artefacts into our simulation that arise from differences in scale reductions of different network parameters. We conjecture that it is possible to downscale a network comprising a few millions of end hosts and link speeds of 10 Gbit/s (i. e., a large AS and the end hosts that generate the traffic routed through it) by a factor of 100 down to speeds 100 Mbit/s and several thousands of end hosts, without suffering from drastic distortions of the workload characteristics. However, our experience with such very large simulations is that they take a very long time to run (one week to one month). Further downscaling of the number of packets flowing through the network is thus desirable.

Naïvely, one might assume that, for downscaling a network by some factor x, it were sufficient to reduce the link capacities and the number of workload-generating end hosts by the same factor x. However, this only works to a certain extent, since it distorts the relationships between important network characteristics such as propagation delay,

queueing delay, transport delay, queue capacities, packet sizes, and TCP window sizes. We now examine these aspects in greater detail and point out observations that are valid for large network simulations in general.

Obviously, reducing the bandwidth increases the transport delay, whereas the propagation delay remains constant, as it only depends on the speed of light in the glass fibre. At bitrates of 10 Mbit/s, the transport delay amounts to 1.2 ms already, and thus comes close to the range of typical propagation delays on long distance links: $300{,}000\,\frac{\text{km}}{\text{s}} \cdot \frac{1}{1.5} \cdot 1.2\,\text{ms} = 240\,\text{km}$ (note that the speed of light in fibre is about 1.5 times slower than in vacuum [Pop]). Thus, further simple bandwidth downscaling increasingly distorts the delay characteristics of the network topology: It reduces the influence of the differences in delay that arise from the differences in physical distance, and increases the differences in delay that arise from network hop distance.

Another factor is the queueing delay: In order to not to let it grow, the router queue sizes need to be scaled down as well. At 10 Gbit/s, a queue size of 2 MBytes can hold 1333 full-size packets of 1500 Bytes (assuming that at the time of this writing, Jumbo frames are not yet widespread outside local area networks). It delays the sending of a packet by 1.6 ms when completely filled, a value that is in the order of the propagation delays, as we have just seen in the transmission delay example above. To retain the original relationship between propagation delay and queueing delay, we have to reduce the queue capacity to 2 kBytes—however, a queue this small will not be able to hold more than just one single full-size IP packet. If we were to use such small queues to offset the aforementioned queueing delay issue, this severely distorts the queueing characteristics of the network, presumably causing a drastic increase in packet drops.

A solution that can ease both of these problems is to reduce the maximum segment size: This way, we can ensure that the queue can hold more packets and, at the same time, we reduce the maximum transport delay. However, applying a scaling factor of 1000 or more is not feasible, since this implies that we have to deal with many tiny packet sizes: zero, one, and maybe two bytes. Thus a solution in the middle needs to be found.

A second reason why we cannot arbitrarily scale down the MTU are the packet headers: By simply reducing the MTU, we reduce the difference between the smallest-possible and largest-possible packet sizes. In order to keep the ratio between header bytes and workload bytes constant, one has to also reduce the IP and TCP header sizes. Alas, our version of SSFNet proved to be unstable when we tried this, and using another simulator was out of question because a re-implementation of REPLEX in a different network simulator, including a faithful reproduction of the worklad generators, the routing subsystem etc., did not seem worthwhile.

Another issue with scaling the MTU are the implications that arise from the subsequent alteration of TCP, HTTP and workload model characteristics. First, downscaling the MTU naturally implies that the TCP MSS is reduced as well, which in turn leads to HTTP headers and HTTP workload to be spread across more packets than in the normal case (unless we reduce their values as well). Second, reducing the MTU increases the number of segments that the TCP window holds. If we want to keep this value constant, we have to downscale the TCP windows as well.

An alternative solution to address the delay-related issues that we mentioned previously is to scale the propagation delays as well: If queueing delay and transport delay are increased by a factor of x, we could simply increase the propagation delays by the same factor x in our simulations. This scales the total delays in the entire network by a factor of x.

However, this simple approach can lead to severe distortions of the TCP workload characteristics, if we maintain our goal of reducing the number of end hosts in our simulated network. First, let us note that the maximum throughput R attained by a (long-lasting) TCP connection with bulk traffic can be expected to be roughly linear with $R \propto M/T$, where M is the MSS and T is the RTT (see, e. g., [HDA05], for more sophisticated TCP throughput estimations). By increasing the delays in the network by a factor of x, we scale T to $T' := x \cdot T$, and thus obtain a downscaled maximum throughput $R' := R/x$. This may seem fine at first glance, as we also reduced the link capacities by a factor of $1/x$. However, bear in mind that our reason for reducing the link capacities is the result of our intention to be able to reduce the number of end hosts and thus the number of packets that we need to simulate. Assuming (without loss of generality) that every simulated host handles roughly the same number of TCP connections and that the network is not heavily overloaded, it is alright to assume that the number of workload bytes B which an average client injects into the network during a certain time span is linear with R (formally, $B \propto R$). Given n end hosts, we thus have a total traffic demand of $n \cdot B$ in our network. Now recall that the reduction of link capacities has to do with the fact that we want to be able to downscale the number of end hosts by the same factor. But if we scale the number of end hosts using the same factor x down to $n' := n/x$, as it was our initial intention, then we have a downscaled traffic demand of $B' := \frac{n}{x} \cdot R' = \frac{n}{x} \cdot \frac{R}{x} \propto \frac{1}{x^2}$. Here, note that $\frac{nR}{x^2}$ is dependent on x^2 and not on x, so this obviously is not linear with our desired reduction factor of x any more! In other words, our downscaled network can be expected to either generate less traffic than it should, or at least to generate traffic with different load characteristics. The obvious way to avoid this is to not reduce the number of simulated clients in the network, but to use quantities as seen in the real world (i. e., hundreds of thousands or millions of end hosts). But this unfortunately does not reduce the complexity and thus the memory consumption of the simulation, although the reduced number of simulated packets at least reduces the CPU time.

Another downside of changing the overall network delays on a large scale is the fact that this also requires us to adjust the timeout distributions of the workload model accordingly—otherwise, the behaviour of the simulated users is not consistent with the timing performance of the network. Our thus increased inter-request times, inter-session times etc. of the workload model, in turn, also reduce the average bandwidth consumed by each end host. This even aggravates the TCP issues discussed in above paragraph. In short, if our downscaling of the network comprises significant increases in delays, our workload hosts can be expected to generate significantly less traffic per host.

To summarise, we can see that there is no easy way to simply reduce the amount of the simulated network traffic (and thus the CPU time consumed by the simulation) such that *all* relevant characteristics of traffic in real-world high-bandwidth networks are faithfully preserved. Simply dividing the link bandwidths and the number of end hosts by the same factor only can work reasonably work well up to a certain scale. If an even larger reduction is required, the most important characteristics to be retained have to be selected, and compromises have to be made on other aspects of the simulation: Apart from link bandwidths and the number of end hosts, other factors that need to be considered are queue lengths and maximum packet sizes; additional tuning options are the various header sizes and file size distributions, as well as the propagation delays of the topology, and the timeouts of the workload model.

Given all these considerations, which of the options mentioned above should we use in our simulations, so that we do not compromise our goal of generating realistic workload traffic? We note that, if we want to simulate a modern network with link speeds of 10 Gbit/s and above, then our simulated network should feature link speeds that are at least 10 Mbit/s—otherwise, we may encounter too large distortions that are initially caused by the transport and queueing delays growing too large, and we would have to resort to further artificial tuning, thereby causing other undesired artefacts. Constraining ourselves to a certain link speed brings about the question of how many workload hosts we are to simulate. Since our workload model is fixed, we determine the average traffic demand for one workload client through simple simulation in a 10 Mbit/s environment under typical RTTs of 2 to 200 ms between client and server and different values from 2 to 200 kBytes for the router queues for a MTU of 1500 Bytes. We find that, in order to achieve high but not extremely high link utilisation with acceptable packet loss rates, a value between 20 and 100 is a good choice for the number of clients, while 8 kBytes to 32 kBytes are good values for the router interface queues. Setting the TCP send and receive windows to 32 kBytes yields sufficiently bursty traffic; using larger values increases burstiness even further but, due to some SSFNet bug, increases the computation time for the simulations.

7.1.6 REPLEX setup

Depending on the size of the topology, we end the startup period after 2 000. . . 30,000 s, and start the *evaluation period*. To simplify our notation, we completely ignore the startup phase and define the beginning of the evaluation period as $t = 0$ for the remainder of the document. The comparison phase (i. e., the time interval during which our algorithm is not active yet) thus is characterised by times $t < 0$. Note that, for the sake of simplicity, we will simply ignore the startup phase that precedes the comparison phase.

The evaluation period is used to study the behaviour of our TE algorithm. Accordingly, the REPLEX instances at each router are now started. Similar to the Web clients, the TE instances start at random times to avoid synchronisation effects. This happens during $t \in [0, T]$ s distributed uniformly across the interval. Communication between

routers has already been enabled at this point. Initially, all route weight settings are neutral; this guarantees an equal traffic distribution among the available equal-cost routes and is the default in real-world networks.

When the algorithm starts, the route weights are still at their unbiased uniform setting (i. e., each inversely proportional to the number of same-cost routes to its destination; e. g., 50:50 in the case of two route alternatives), and have not yet been optimised. Thus the time at which the TE instances are started can be seen as a situation in which a drastic large-scale traffic change has suddenly caused (almost) all weights in the entire network to be maladjusted—which our algorithm now needs to mend.

We mentioned earlier that REPLEX allows to optimise different metrics. In our simulations, we use the following metrics to optimise (i. e., we use REPLEX to minimise them):

Link load: This is presumably the most popular metric with network operators. Most TE schemes strive to minimise the maximum link load in the network, with the idea that higher link loads feature increased congestion and thus a deterioration of service quality.

Nonlinear link load: Links that feature a high link load are much more likely to suffer from packet losses than links with only a medium load; i. e., packet losses grow faster than linear with link load. Similar to other TE schemes, e. g., [FT00], we therefore also try an approach where we valuate the link load with a nonlinear utility function. In Section 6.3.1 we argue that linear latency functions[1] allow us to find the (global) optimum. If we now consider that the actual goal is not to minimise link loads, but rather to minimise packet loss probabilities, then it makes sense to choose a nonlinear utility function that accurately captures the relation between link load and packet losses.

Queueing theory suggests for simple M/M/1 queueing models that the increase in latency is hyperbolic with the load [GH98]. Although we cannot assume that queues in real networks have exponentially distributed packet interarrival times due to the self-similarity of the network traffic characteristics [LTWW94, CB97, CB96, WPT98, Fel00b, PW00], it nevertheless is an indication for a good starting point to minimise $\frac{1}{\textit{remaining capacity}}$, i. e., $\frac{1}{1-\textit{link load}}$. This implies that our utility function approaches ∞ as the link load approaches 1. However, it is possible that the link load actually approaches or even exceeds this value. A valuation of ∞ for a link results in immediate drawback of all traffic from this link, which is certainly an unfavourable overreaction. To avoid this happening, we add a small additional factor κ and thus use

$$\textit{nonlinear link load} := \frac{1}{1 - \textit{link load} + \kappa}$$

[1] Let us remark again that here, the term *latency* is used in its meaning for theoretical computer science, i. e., as a generalised cost measure, but not in the meaning of actual queueing, transmission, or propagation delay.

as our formula for deriving the nonlinear link load from the measured link load. We use $\kappa = 0.2$ in most simulations involving this metric, as our simulation results suggest that even a heavily overloaded link rarely exceeds a load of 1.1.

Packet loss probability (loss rate): Instead of trying to model the packet losses as a function of the link load, we may as well use the probability of drop-induced packet losses itself as a metric. We note that this is a potentially dangerous approach, as the congestion control mechanism of TCP strives to minimise this metric as well—thus if we are to use packet loss probability as the metric for REPLEX, we will not only have two interacting control loops (i. e., REPLEX and TCP congestion control), but even two interacting control loops that try to optimise the same objective.

Another aspect that we need to consider is the propagation of the packet losses via our distance vector like protocol. Recall that, for the link load, we take the maximum of our own measured link load, and the maximum of the link load on the further route, as reported by our neighbour. Although this makes sense for link loads, it does not make much sense for packet losses; certainly, a route featuring one link with loss probability 1 % (and no further losses beyond that link) is preferable over another route with three links, each of with with a loss probability of 0.5 %: In the end, the latter has a higher overall loss probability.

But just as we cannot drill a hole of 10 mm in diameter using one 6 mm and one 4 mm drill,[2] we cannot simply add the packet loss probability of a link to the loss probability of the downstream route—rather, we have to combine both to obtain a meaningful value. Let p_i be the probability that a packet gets dropped at the i-th link of a route, then the probability $\bar{p}_i$ for the packet to pass link i unharmed is $\bar{p}_i = 1 - p_i$. Hence, the probability for the packet to safely reach the end point of a route with n links is $\prod_{i=1}^{n} \bar{p}_i$. Suppose that a packet already has reached the k-th link of the route, then the probability $\bar{\mathcal{P}}_k$ for it to safely travel along the remaining route without being dropped is $\prod_{i=k}^{n} \bar{p}_i$. Now note that this is actually $\bar{p}_k \cdot \bar{\mathcal{P}}_{k+1}$. We thus can calculate the packet loss probability $\mathcal{P}_k = 1 - \bar{\mathcal{P}}_k$ on the remaining route using our distance-vector approach as $\mathcal{P}_k = 1 - \left(\bar{p}_k \cdot \bar{\mathcal{P}}_{k+1}\right)$, i. e., in the end we obtain

$$\mathcal{P}_k = 1 - \left((1 - p_k) \cdot (1 - \mathcal{P}_{k+1})\right),$$

where p_k is our own measurement of packet losses on link k, and $\mathcal{P}_{k+1}$ is the packet loss report that we received from the neighbour at link k. We use $\mathcal{P}_k$ in our REPLEX calculations, and also use it (averaged together with further $\mathcal{P}'_k$, $\mathcal{P}''_k$, $\mathcal{P}'''_k$,… of alternative routes, weighted by their respective weights) in our report that we send to our upstream neighbour $k - 1$.

[2] Remember the footnote on page 7…

7.2 Measuring Performance

So far we have been talking about the setup of the simulations that we will use to measure REPLEX's performance—but how do we actually *do* measure its performance?

Generally speaking, we examine the performance under two criteria: First, how much does applying REPLEX increase the "performance" of the network (note that we need to define "performance" before we can answer this question); and second, how fast does REPLEX achieve this goal? This opens two questions: how do we define the "performance" of the network, and how do we define the speed of convergence, i. e., how do we define "convergence"?

7.2.1 Network performance

A number of criteria can be thought of as performance measures. Since a common optimisation goal in traditional (i. e., offline) traffic engineering is to minimise the *maximum link load* on any link in the network, this measure is naturally an interesting performance indicator.

maximum link load

One of the reasons to use this goal in traditional TE is that a high load on a link increases the probability for packets to be dropped when they are to be sent over it. An increased loss rate, however, does not only decrease the quality of service as perceived by the end users—it can, moreover, lead to synchronisation effects across otherwise independent TCP connections, since all connections affected by the same packet loss burst will react with their congestion control mechanisms (slow start and/or fast retransmit) around the same time. This synchronisation, in combination with the effectively non-linear increase of the slow-start phase, will increase the traffic's burstiness and thereby deteriorate the service quality even more. Therefore, the *packet loss rate* is another useful measure for network performance.

packet loss rate

Another measure that is of interest for the network operator is the number of bytes that are transmitted through the network. Greatly oversimplifying economic details, we can generally assume that the financial revenues increase as the number of transmitted data increases. On the other hand, we can assume that TCP congestion control will make use of additional network capacities, thereby shortening download times and thus allowing more downloads per time unit.[3] Therefore, the number of transmitted IP payload bytes per time unit, i. e., the *network throughput*, is yet another useful measure for network performance.

network throughput

Apart from the packet loss rates, the criteria mentioned so far view performance from the position of a network operator, i. e., at the IP level. However, it is not the network operator but the end users who utilise the network and who pay for it; so measures that describe the performance as experienced by the end user are also good performance indicators. Since TCP is used solely as our workload, it is thus useful to measure the *TCP goodput*, i. e., the total number of TCP payload bytes that were actually received by

TCP goodput

[3] The inter-request and inter-session times in our workload model are counted from the *end* of a request/session to the *beginning* of the next one. Slow download rates will thus delay the downloads of further objects, thereby reducing the overall number of bytes that one client downloads.

the end hosts per time interval. If faster download rates increase the total number of bytes in the network, then the TCP goodput should increase likewise (unless the TCP connections suffer from heavy retransmits induced by massive packet reordering or packet losses).

We mentioned earlier that, due to the hashing approach, REPLEX ensures that packets pertaining to one TCP connection always travel along the same path, since packet reordering negatively affects TCP performance. However, a few packets actually *may* be reordered when REPLEX changes the weights, i. e., assigns some hash bins (and thus, potentially, some active TCP connections) to a new route. As the receiving TCP stack is not clairvoyant, it cannot know that a missing packet has just been reordered and will thus arrive later than its successor; therefore it does not distinguish between TCP segments being reordered and TCP segments being lost. To quantify the extent of possible REPLEX-induced reorderings, we measure the *TCP packet loss rate* per time interval, i. e., the number of all TCP segment losses as perceived by the receiving end hosts (be they caused through congestion or through packet reordering), divided through the total number of TCP segments injected into the network during the time interval under consideration.

TCP packet loss rate

retransmission ratio

Another interesting performance indicator is the *retransmission ratio*. Whenever a TCP receiver deems a segment as having been lost and indicates this to the sender, then the latter has to retransmit the segment in question. By relating the number of TCP segments that have to be retransmitted to the absolute number of TCP segments, we can judge the efficiency that TCP can achieve in the underlying network. The retransmission ratio naturally includes both retransmits caused by packet reorderings, as well as retransmits caused by packet losses due to congested links.

7.2.2 Determining convergence

So far, we have talked about how to measure the effect of REPLEX on a network's performance. However, since REPLEX is a dynamic traffic engineering algorithm, it is not only of interest how *much* we can gain by using REPLEX, but also how *fast* REPLEX can attain an improved performance level. In other words, we need to measure its speed of convergence.

As we shall see from the experimental results in, e. g., Section 7.3, we find that some of our performance measures seem to reach their optimal level quicker than others—for example, as soon as we enable REPLEX, the number of transmitted bytes increases much faster than the route weights do. Therefore, although it may at first sight feel natural to define convergence solely on the output of the algorithm, i. e., the changes to the weights, it is also interesting to see how quickly REPLEX stabilises at a level with increased network performance.

Since we use a statistical workload model, we have significant fluctuations in our network traffic even if we do not employ REPLEX at all. Therefore, we cannot expect any of our performance measures to monotonically increase over time and finally converge to one fixed value. Rather, it is natural for our performance measures to fluctuate—not only before and after REPLEX has converged, but even during the convergence process.

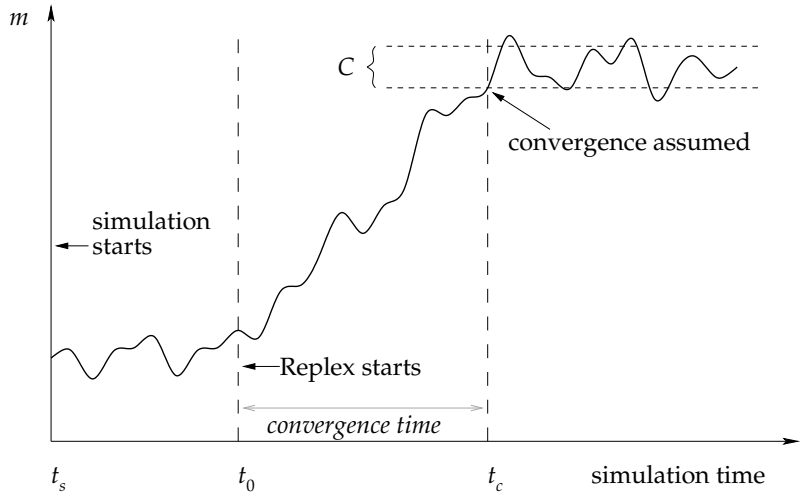


Figure 7.3: Calculating the convergence time t_c, based on a sequence of measurements m_i.

This, of course, opens the question of how one can determine when a series of measurements has stabilised (converged), and when it has not yet reached a stable level but is still in the process of converging.

Assume that we want to determine the convergence time t_c of the time series that represents our measurement value m (e. g., m can be the weights that REPLEX adjusts). Formally, we have measurements $m_i, i \in 0, \ldots, e$ where $i = 0$ corresponds to the time at which we start REPLEX, and $i = e$ corresponds to the end of our simulation time. We want to find some c such that all $m_i, i \geq c$ have "converged", i. e., have become stationary and remain within some "convergence region". We call the time t_c that corresponds to the measurement m_c the *convergence time*, supposing that $t_0 = 0$ corresponds to measurement m_0.

convergence

Let us initially suppose that our simulation runs for an infinite time after the start of REPLEX, i. e., $e = \infty$. In this case, we can calculate a *statistical convergence time* t_c in the following way. Since we have infinitely many m_i spanning an infinite amount of simulation time, we can neglect the initial convergence phase $m_0, \cdots, m_{c-1}$ (since it is finite) and hence assume the m_i values to be stationary. Therefore, we can calculate the expected value of m (which in this case exactly coincides with the arithmetic mean) and other descriptive statistics. More specifically, we can calculate some kind of *convergence corridor* C, within which the majority of the m_i fall, using some kind of *corridor function* f. For the time being, imagine f to be something like $f(m_1, \ldots) = \mu(m_1, \ldots) \pm 1 \cdot \sigma(m_1, \ldots)$ or $f(m_1 \ldots) = (\text{first quartile}(m_1, \ldots), \text{third quartile}(m_1, \ldots))$. After having defined our convergence corridor, we can define convergence to occur at the first measurement m_c that falls into our convergence corridor. Formally, $m_c \in C \wedge \nexists i < c$ such that $m_i \in C$. See Fig. 7.3 for a graphical explanation. Note that *some* subsequent $m_i, i > c$ may fall outside the corridor C again, as it is shown in the figure. For example, this is naturally the case if we use quantiles as our corridor function.

convergence corridor
corridor function

Of course, our initial assumption of having an infinitely-long running simulation is not realistic; in fact, we only have a limited number of measurements, i. e., $e << \infty$. If we have a large number of measurements, we can still calculate the corridor function using all m_i—however, then our argument that the measurements pertaining to the initial convergence phase do not affect the extent of our corridor does not hold any longer. In an extreme case, m_c occurs just shortly before m_e (i. e., the end of the simulation), and thus most m_i fall into the convergence phase (i. e., most $i < c$). Therefore, they must not be used in the calculation of the corridor, since we want our corridor to describe the location of typical m_i values *after* convergence has occurred; we therefore can only use those measurements $m_i, i \geq c$ that we obtain *after* the convergence time t_c to calculate our corridor C. In other words, to calculate C, we need to know t_c—but recall that our reason for calculating C is that we need it in order to compute t_c. Is the cat biting its own tail?

We can resolve this cyclic redundancy by assuming that those $m_{b\ldots e}$ (with b only a little smaller than e) that appear near the end of the simulation almost certainly belong to the converged state. We calculate C based just on these $m_b, \ldots, m_e$. Then we can check if all prior $m_i, i < b$ are indeed outside our calculated C. If this is the case, we have, by definition of c, found that $c = b$ and thus determined the convergence time (assuming without loss of generality that m_b is the first measurement that lies within C). If this is, however, not the case—i. e., there is at least one m_i with $i < b$ such that $m_i \in C$—, we can iteratively reduce b and check again. This way, we check at which earlier point in time the convergence region is reached for the first time. Of course, reducing b also means that we include more and more m_i in our calculation of C.

To summarise, we grow the region which we assume to be converged from the end of the simulation, thereby constantly adjusting the convergence corridor, until we reach the first point of the converged phase by our definition of convergence.

We calculate the point after which we consider our series of measurements to be converged using Algorithm 6. In addition to the symbols introduced so far, the algorithm uses the following additional variables: m^{conv} is the set of measurements that are considered to be converged as by our definition, i. e., $m^{conv} := m[c, \ldots, e]$. $m^?$ is the current candidate for m^{conv}, which we test if it fulfils our convergence criterion (i. e., it more or less corresponds to the variable m_b in the description above). We label its convergence corridor with $C^?$. As can be seen, Algorithm 6 performs exactly the computations that we described above, and thus yields the convergence region according to our definition.

Note that this simplified version does not consider the case that some of the $m_i, i < c$ may lie below corridor C while others may be located above it. It is an interesting question whether a subsequence of measurements that are intermittently either above or below a convergence corridor can be regarded as being converged or not: In those cases where convergence expresses itself as a decrease in the amplitude of fluctuations, it makes sense not to consider such a subsequence as converged. In other cases where convergence appears as level changes (for example, from low to high values), it can make sense to neglect one of the boundaries of the corridor (in the example case, the upper boundary). See Fig. 7.4 for a graphical explanation of the two possibilities. The measurements involved in our REPLEX simulations mostly take either variant of the

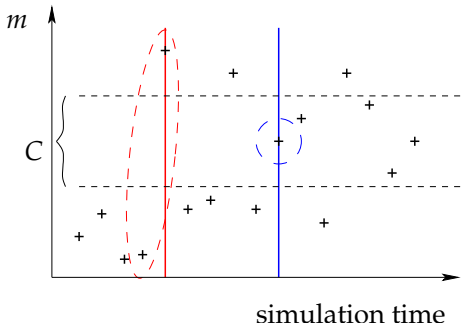


Figure 7.4: Possible definitions for the convergence condition can be either the first time that the convergence corridor C has been crossed (red, left) or the first time that some measurement m_i lies within the boundaries of C (blue, right). We choose the latter, stricter approach.

Algorithm 6 Calculating the statistical convergence of a time series.

Require: A time series of measurements $m[1, \ldots, e]$
1: $m^{\text{conv}} \leftarrow m\,[e-2, \ldots, e]$
2: $C \leftarrow f(m^{\text{conv}})$
3: **for** i **in** $3, \ldots, e$ **do**
4: $\quad m^{?} \leftarrow m\,[e-i+1, \ldots, e]$
5: $\quad C^{?} \leftarrow f(m^{?})$
6: $\quad$ **if** $m^{?}\,[1]$ inside $C^{?}$ **then**
7: $\quad\quad m^{\text{conv}} \leftarrow m^{?}$
8: $\quad\quad C \leftarrow C^{?}$
9: $\quad$ **end if**
10: **end for**

second form, i. e., convergence towards a level higher or lower than the one prior to the actions of REPLEX. Manual inspection of the data reveals, however, that intermittent over- and undercrossing of the corridor without any further measurement actually falling within the corridor occurs only very infrequently and, if at all, only for very short periods of time. We thus restrict ourselves to the stricter and more conservative approach of requiring the first measurement of the converged set to stay within the corridor boundaries (depicted in blue in Fig. 7.4), thereby possibly obtaining slightly longer convergence times.

So far we have disregarded the range of m_i prior to the start of REPLEX, i. e., $m_{-1}, m_{-2}, m_{-3}, \ldots$. In an extreme case, our algorithm may have no effect on m at all—then, most of the m_i for $i < 0$ are inside C, as well as most $m_j for j > c$; and those few $m_0, \ldots, m_{c-1}$ that were outside of C happen to be just some statistical fluctuation. In such a case, the value c is meaningless. Unfortunately, this is still the case if only *some* of the m_i, $i < 0$ fall within C, even if running REPLEX otherwise does have some effect on m. Note that it is not sufficient to compute a "pre-REPLEX" corridor $\bar{C} = f(m_{-1}, m_{-2}, \ldots, m_{\text{start}})$ and to prove that $\bar{C} \cap C = \{\}$. Since we only have a finite number of measurements m_i for

$i < 0$, we can however check that none of the pre-REPLEX measurements m_i for $i < 0$ lies within C. If this is the case, we take this as a strong indication that the typical levels with and without REPLEX are separated enough so that our convergence calculation is sensible. Otherwise, we cannot be sure whenever an m_i is within C, if this is really due to REPLEX's action, or whether it is actually just some statistical fluctuation that may well have happened without any REPLEX influence at all.

To summarise, in this case we are not able to make any claim with regards to convergence time using this methodology.

7.2.3 Measuring weight changes

In the previous section, we have presented a method for determining the point at which a time series of measurements has converged—but *what kind* of measurements do we actually want to check for convergence? As pointed out previously, we have two large groups of metrics to choose from: On the one hand, REPLEX's immediate output, and on the other hand, network performance metrics that are affected by REPLEX's actions.

The first group, i. e., the output of our algorithm, contains the weights and weight changes. In small and primitive test scenarios that feature only extremely few equal-cost routes, we can measure convergence times for each individual route. In the simplest case, i. e., having only two equal-cost alternatives to the same destination network, the weights for the two routes are complementary to each other and thus convergence occurs at the same point in time for both (unless we made an unwise choice regarding to our corridor function f).

In more complex scenarios that contain many equal-cost routes differing in length and number of equal-cost alternatives, we can calculate the convergence time for each individual route. Although we cannot obtain "one" generalised convergence time for our algorithm this way, we can use these values to obtain a histogram, or to calculate descriptive statistics (minimum, maximum, average, median) for the convergence times, and thus get an impression of typical convergence times.

Alternatively, we can sum up all weight changes that occur within one time interval. At the start of REPLEX, we expect the weight changes to occur with a large amplitude: At this point in time, the weights are completely un-tuned towards the actual traffic demands in the network, and REPLEX is constructed such that it applies larger weight changes if more gain is to be expected, whereas it applies only small changes if less gain is to be expected. The latter, of course, should be the case after some time has elapsed and the first weight shifts already have led to a significant increase in network performance.

We can thus use the accumulated weight changes per time unit for another purpose. Recall that TCP performance is negatively affected if packets of the same TCP connection overtake each other, which was our motivation for using our hash-based approach instead of random-based statistical multiplexing of packets. Note, however, that packet reordering may still occur when an actively used TCP connection is suddenly sent along a different path, due to a weight change by REPLEX. If a larger weight shift occurs at the

same time, then each from a potentially larger number of TCP connections may suffer from their packets being reordered. As the reorderings happen at the same time, this bears the danger that the congestion control mechanisms of all affected TCP connections become synchronised—which by the resulting amplifying effect increases traffic burstiness and thus can have a further negative impact on network performance. It is therefore desirable to make small weight changes, in order to keep the number of simultaneously affected TCP connections small. Obviously, it thus makes sense to "punish" heavy increases or decreases in weight. Hence it is better not to sum up the weight changes per time unit, but to sum up the *squared weight changes per time unit* as a means to capture REPLEX's activities.

squared weight changes

7.3 Artificial topologies

REPLEX's theoretical analysis indicates fast convergence. But as the theoretical analysis cannot take real world factors into account, e. g., bursty traffic, or interactions with the feedback loop of TCP congestion control, we now evaluate its performance in a network simulator under realistic workload conditions. We start the evaluation with small and simple topologies. This helps us to tune the various parameters of the algorithm, and to evaluate its impact on the network performance.

7.3.1 Experiment planning

The the many parameters of our algorithm raise the question about how to choose the best approach to find good parameter settings—simply examining all possible combinations of three different values for each parameter ($\alpha, \beta, \epsilon, \eta, \lambda, T$, and the metric that we want REPLEX to optimise), and using 10 simulation runs for each combination (so as to reduce the influence of random statistical fluctuations) results in an experiment table of size $10 \cdot 3^8 = 65610$ experiments. As each of these experiments typically runs for 15 minutes at the very least (many of them require 30...300 minutes), this would require roughly two years of CPU time, and probably more. Therefore, we have to find a more effective approach.

Initial experiments in the parameter space [FKF06] indicate that the most influential parameters on REPLEX's convergence time are λ (adaptation speed) and T (decision interval), followed by η (factor for the exponential moving average), while the other parameters' influence is less pronounced. We therefore devise the following experiment plan:

1. First, we determine good choices for λ for our three different metrics in a very simple topology, with "reasonable" standard values for the other parameters, focusing on just one REPLEX instance.

2. Having found good choices for λ and the REPLEX metric, we then analyse the influence of η. Note that in contrast to most other parameters, the exponential

moving average affected by η does not only influence the decisions of one REPLEX instance; its value also affect the updates that are broadcast to neighbouring instances. Therefore we analyse the influence of η in topologies with different-length "chains" of REPLEX instances.

3. One of the key questions in load-adaptive traffic engineering is the question regarding the time scale on which we can react to traffic fluctuations without running into negative effects, i. e., oscillations caused by overreactions or by too strong interaction with the TCP congestion control loop. In other words, it is important to see how small we can make T. To analyse this, we use a very simple setup without a bottleneck link, but that features routes with different latencies. We use this to analyse the extent of packet reordering caused by REPLEX rerouting actively transmitting TCP connections, and its effects on TCP performance. We combine different settings of T with different settings of λ and η.

4. In the previous chapter, we argue that theory gives a strong indication that a set of competing REPLEX instances converges to a stable solution. To analyse how well REPLEX performs under realistic set-ups, we analyse a small scenario where several REPLEX instances compete over a bottleneck link. We use this set-up to also show that our previous results based on just one deciding REPLEX instance can be generalised for scenarios involving multiple instances.

5. After having found a good parameter setup, we finally analyse REPLEX's performance in setups involving more complex and realistic topologies. Furthermore, we compare the performance improvements that we can achieve using REPLEX to those that are achievable using traditional traffic engineering.

7.3.2 The influence of the rerouting speed λ

To avoid oscillation effects, we have to judiciously choose the amount of flow λ that the router can shift during one decision interval. In order to make efficient use of CPU time, we restrict ourselves to a very simple topology where we can focus on just one router. Apart from varying λ, we keep the other parameters at default settings of $T = 1\,\mathrm{s}$ (which it significantly larger than the client-server RTT and thus allows TCP congestion control to converge before the next REPLEX decision), and $\alpha = 0.25$, $\beta = 0.1$, $\eta = 0.5$, $\epsilon = 0.0$, which we found out to be good choices in prior, preliminary experiments (partially presented in [FKF06]). As the optimisation metric, we initially choose the link load, as it is the traditional optimisation goal in traffic engineering.

We perform our simulations for finding good values for λ using the topology depicted in Fig. 7.5. It consists of six routers, a server cloud at $R3$ (source), and a client cloud at $R1$ (sink). The client cloud consists of 80 HTTP clients; the server cloud hosts two HTTP servers. Unless otherwise stated, the link bandwidths are 10 Mbit/s, the interface queue lengths are 16 kBytes, and the propagation delay on each link is 10 ms. The underlying routing setup is done by the OSPFv2 implementation for SSFNet [Bö03],

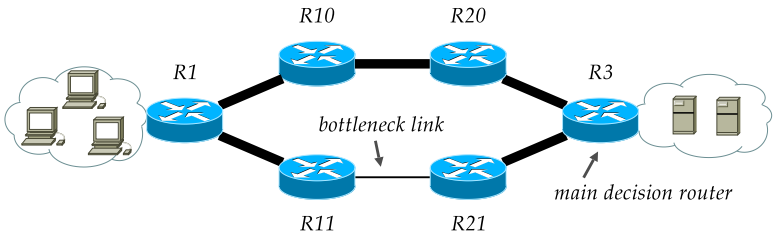


Figure 7.5: Standard topology for many of the experiments with an artificial topology.

using uniform weights. This means that the shortest paths used for routing are solely based on hop distance and not on any other factors.

In Section 7.1.5 we argue that 10 Mbit/s is a very small value if we are to extrapolate our simulation results to today's real-world networks with link speeds of 10 Gbit/s. However, due to the simplicity of the topology, scale interactions as described in Section 7.1.5 are not really an issue. On the other hand, using a low link speed reduces the number of packets that need to be simulated per time interval, which help save computation time—which really is an issue, since the bursty workload traffic forces us to run each simulation multiple times, so as to reduce the influence of statistical fluctuations on the results.

Each router in the topology runs an instance of REPLEX. However, only routers $R1$ and $R3$ can choose between multipath alternatives for connecting the two workload clouds: They can either route via $R10$ and $R20$, or via $R11$ and $R21$. Therefore, the REPLEX instances at the other routers only perform measurements, and communicate using our DV-like protocol; otherwise, they have no influence on the traffic in the network. Since an HTTP request is typically much smaller in the number of bytes than its corresponding HTTP response, using a client cloud and a server cloud results in an asymmetric traffic ratio where the majority of the traffic is flowing from the right (server cloud at $R3$) to the left (client cloud at $R1$). To investigate the decisions of REPLEX, it therefore suffices to look at the decisions of $R3$, as the decisions of $R1$ on the overall traffic only play a marginal role.

We start our simulations with reducing the bandwidth of the link between $R11$ and $R21$ to 1 Mbit, thereby artificially creating a bottleneck. Neither $R1$ nor $R3$ are aware of this fact, as we do not reflect it through, e. g., OSPF weight settings. The bottleneck is thus only visible to the other routers through the measurements of $R11$ and $R21$, made available by REPLEX update messages with the neighbouring routers. This way, we can examine the effectivity of both the weight adaptation, as well as the communication between REPLEX instances.

In Fig. 7.6, we see that λ does have a significant influence on the time that REPLEX needs for convergence. The top row plot shows the weight that the REPLEX instance assigns to the route going over the non-bottleneck link. Considering that the link speeds are 10 Mbit/s vs. 1 Mbit/s, one can assume the theoretically optimal weight to be $\frac{10}{11}$ (marked with the grey dash-dotted line in the top-row plot). We see that convergence

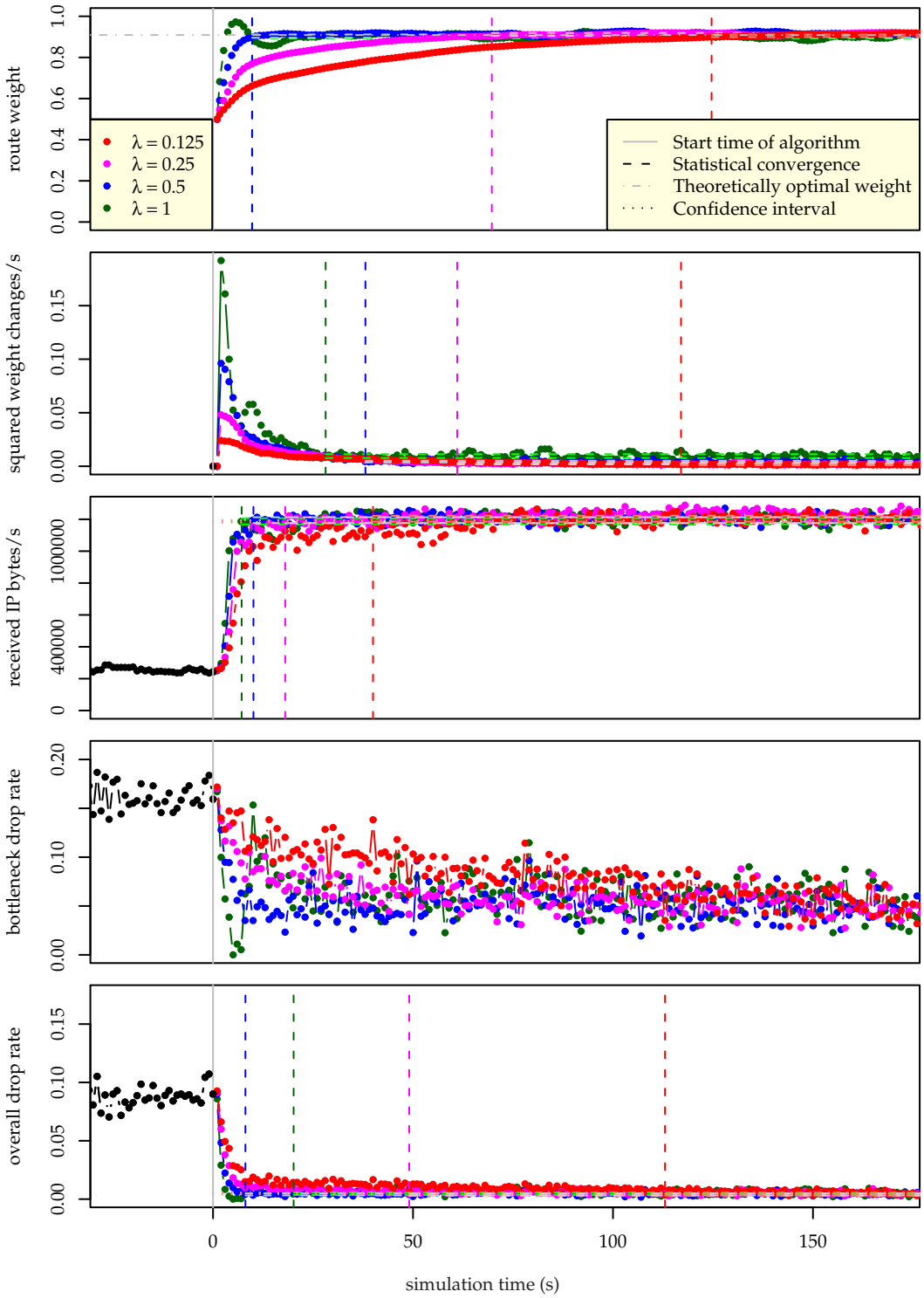


Figure 7.6: REPLEX converges quickly within seconds. λ affects convergence speed, but not final solution (top plot). Optimisation metric is link load. Averages of 10 simulation runs.

(dashed vertical lines) of the route weight can be reached relatively quickly on the order of a few seconds (left, higher lambda) to the order of one or two minutes (right, smaller lambda). Once REPLEX has converged towards a solution, the values stay closely within a tight convergence region for most of the time (dashed horizontal lines; barely discernible). The additional dotted horizontal lines (barely discernible) mark the confidence intervals of the average chosen weight. The fact that the horizontal lines marking the convergence corridors and those marking the confidence intervals are almost impossible to distinguish shows that the converged solution is independent of the choice of λ, just as one expects. We moreover see that the weight increases more or less monotonically, until the converged state is reached. The only exception to this is the setting $\lambda = 1.0$ (green dots), where we can see a brief oscillation that lasts for one single period of about 20 seconds.

The weight changes (second row from top) exhibit a slightly different behaviour: For $\lambda = 0.5$ and $\lambda = 1$, convergence of the weight changes is reached significantly later than the weights themselves. This is an indication that, while the weights may appear to have converged, they are in fact subject to some oscillations (similar to the obvious one that can be seen in the first 20 s for $\lambda = 1$ in the top-row plot, but much smaller in amplitude). For $\lambda < 0.5$, however, convergence of weight changes happens at almost the same time as convergence of the weights themselves takes place.

Interestingly, the workload traffic adjusts to the changing situation even quicker than the weights themselves: The middle plot shows that the IP throughput reaches convergence much faster than the weights (top plot) or weight changes (2nd row). This is due to the fact that the congestion control loops of live TCP connections being re-routed quickly detect the additional capacity of their new route. It is apparent that just a few of these reroutings need to take place before the non-bottleneck link is fully utilised—e. g., compare the top-row and the middle-row plot for the slow-moving $\lambda = 0.125$ (red) around $t = 20$ s. Another fact which supports this is the observation that during the first very few seconds, the traffic rate (unlike the weights in the top-row plot) does not increase very much, before it is subject to an increase that is way steeper than the increase in weight.

The same picture emerges when we view the packet drop rate, which is shown in the last two plots. The first one of them (i. e., fourth row from top) shows the drop rate at the interface that is the entrance to our artificial bottleneck link. Here, we see that the bottleneck drop rate is reduced rather slowly for $\lambda = 0.125$, which corresponds to the slow weight changes. (Note that, due to the overlapping ranges of measurements, it is not feasible to obtain convergence times for this performance metric by using our method from Section 7.2.2 on page 128.) Likewise, the bottom plot shows the overall drop rate, i. e., the sum of the packets dropped either at $R20$–$R10$ or at the bottleneck link $R21$–$R11$, divided by the total number of packets that were sent into the network. In contrast to the bottleneck drops, we see a very swift decrease in overall drops even for $\lambda = 0.125$, whose convergence times more or less correspond to those of the workload bytes (3rd plot from top). This is due to the fact that TCP congestion control effects a quick increase in utilisation on the non-congested link, thereby drastically increasing the number of transmitted packets while the absolute number of drops in the network increases only marginally.

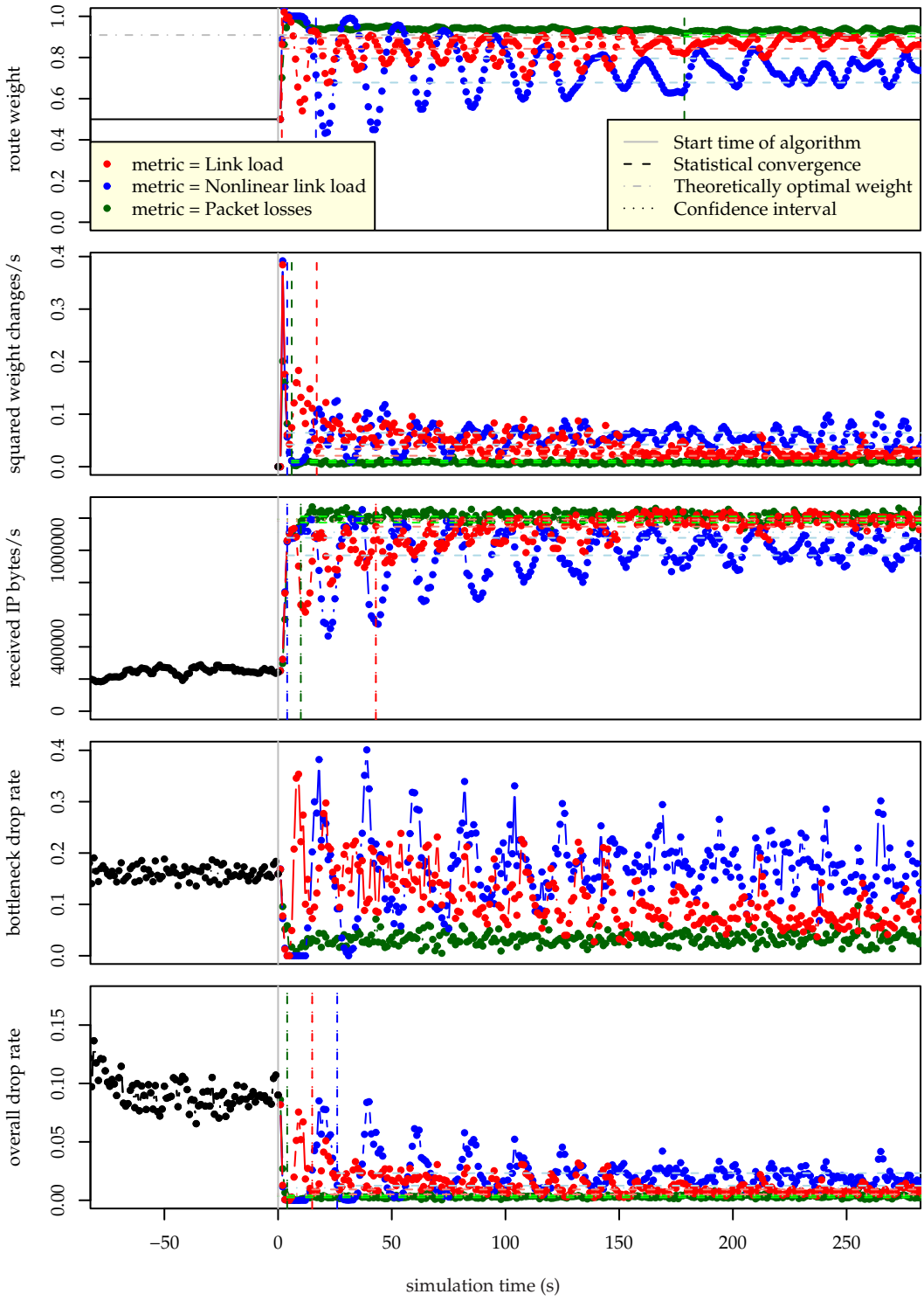


Figure 7.7: Comparison of different metrics at the aggressive setting $\lambda = 2$. Nonlinear route load leads to heavy oscillation. Normal (linear) route load still shows pronounced oscillation. Packet losses show almost no oscillation. Averages of 10 simulation runs.

7.3.3 Which metric should REPLEX optimise?

So far, we have used the link load as our metric to optimise, just as it is normally the case with traditional traffic engineering. We now investigate the results of using other metrics, i. e., the non-linear link load that we discussed in Section 6.4.5, and packet losses. Clearly, one expects different outcomes for the different metrics even if all the other parameters are kept the same, as the weight changes are linear with λ and with the performance difference; and if the performance difference is changed (due to using a different metric) while λ remains the same, then the outcome in weight change naturally will be different.

Figure 7.7 compares the outcome for our three metrics at the rather aggressive setting $\lambda = 2$. The other parameters remain the same as for the simulations from Fig. 7.6. We immediately see that a choice of $\lambda = 2$ results in heavy oscillation of the weights (top plot) and, respectively, weight changes (2nd plot from top) for the nonlinear route load (blue). This heavy oscillation also influences the network's performance (other plots). While the oscillations in weight gradually reduce (top plot), they never disappear entirely over the entire 4000 s of simulation time (plot not shown). In spite of the heavy oscillations, even this obviously bad setting still is able to increase the network throughput in the long run almost as well as in other experiments (middle plot, and comparison to Fig. 7.6). Interestingly, the worsened performance is not only due to the oscillation, but due to the convergence around a weight corridor that does obviously not contain the theoretically optimal weight setting of $\frac{10}{11}$ (top plot; dashed faint blue lines indicate convergence corridor for the weights; dash-dotted grey line shows theoretically optimal weight). While this strategy shifts a large amount of traffic towards the uncongested link, it is apparently not enough to relieve the congested link of its congestion (fourth plot from top), which in turn negatively affects the overall drop rate (bottom plot) when compared to the other simulations. Another fact worth noting is the convergence of the weight changes by definition, which—contrary to the appearance of the plot—is very fast (vertical dashed blue line, second plot from top): The fast "convergence" is due to the fact that the weight changes almost show no real convergence. Therefore, by our definition, the convergence corridor becomes very large, and it is thus entered very quickly, as the values swing back and forth. The same holds for the "convergence" times in the other plots (vertical dashed blue lines).

The situation for the "normal", i. e., linear link load is slightly better (shown in red). Here, we also see significant oscillations in the weights and weight changes (top 2 plots) that influence the network's performance (other plots). However, the weights converge around a corridor that is closer to the theoretical optimum (dashed faint red lines in top plot), which in turn relieves the bottleneck link in the long run (fourth plot from top), thus results in an improved overall loss rate (bottom plot), which in turn increases the network throughput when compared with the nonlinear link load metric (middle plot).

The figure tells clearly that, at $\lambda = 2$, we achieve the best performance if we use the packet drops as our metric to optimise (shown in green): This setup shows a remarkably fast convergence within just a few seconds, and does not exhibit any oscillation behaviour, apart from a one-period small "swing-in" phase that is barely noticeable (top-row plot, first 20 seconds).

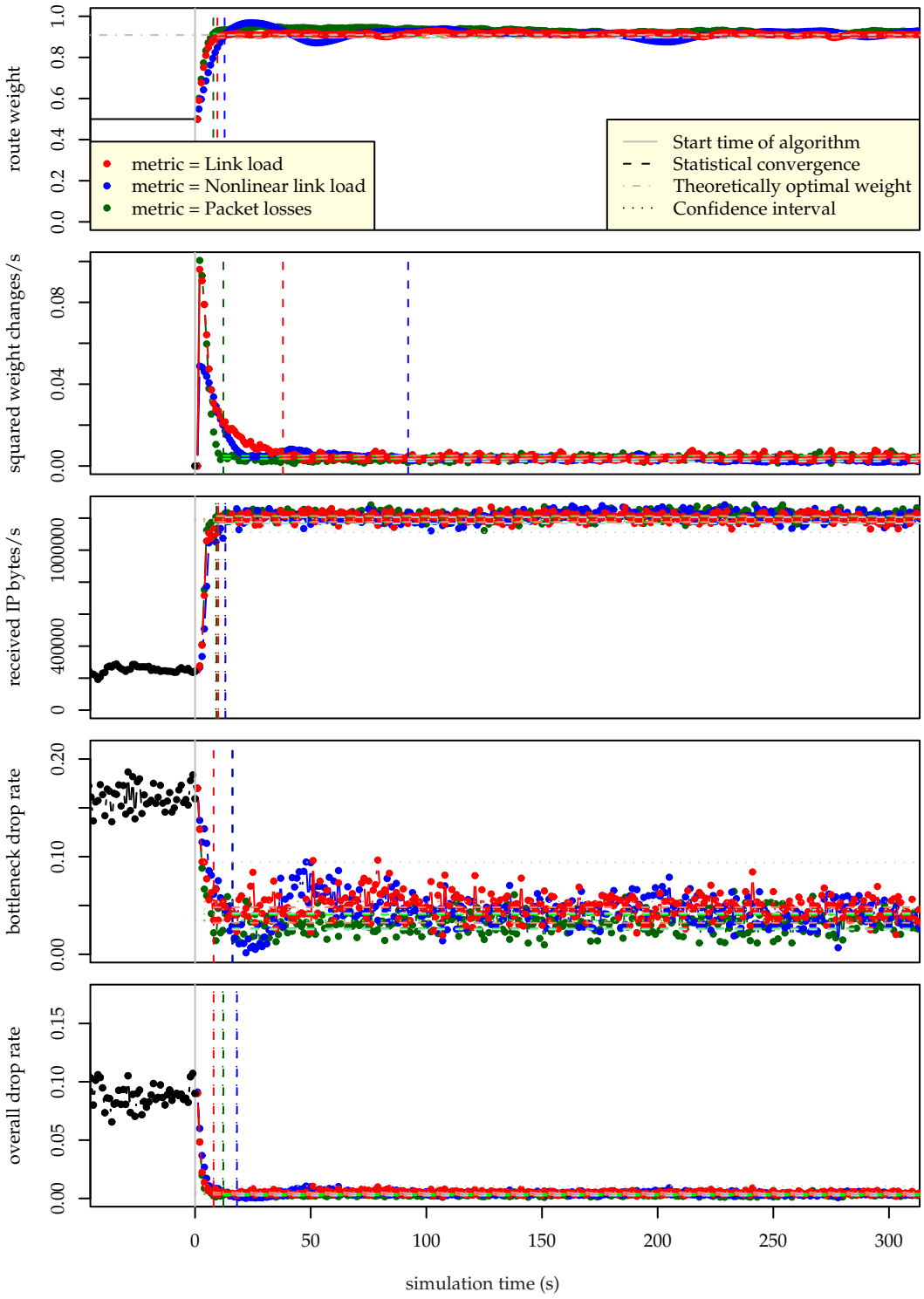


Figure 7.8: Comparison of different metrics at their respective highest values of λ that do not yet lead to (much) oscillation (i. e., "optimum" values for λ). REPLEX with packet losses metric (green) converge fastest; nonlinearly weighted link load (blue) gives worst results. Averages of 10 simulation runs.

Obviously, it does not make sense to compare different metrics while leaving the other parameters untouched. In fact, each metric has its unique optimum setting of λ that allows the fastest possible convergence without going into oscillations. Analysing various settings of λ and, in part, other parameters for the three different metrics, we found the following values of λ to be good choices:

- link load: $\lambda = 0.5$
- nonlinear link load: $\lambda \leq 0.25$
- packet losses: $\lambda = 1$ (in fact, values up to 3 are possible without medium- and long-term oscillation, but will lead to small swing-ins of one or two periods; cf. Fig. 7.9)

We compare these settings in Fig. 7.8. We see that the difference between the three metrics becomes small, albeit notable in some details.

At first sight, convergence is achieved very fast and at almost the same time (top plot; dashed vertical lines indicate convergence). We note a small advantage for packet losses (green) and linear link load (red) over nonlinear link load (blue). This picture remains almost untransformed when we look not only at the weights (top plot), but also at the network performance (middle plot to bottom plot).

However, the convergence in weight changes (2nd plot from top) is achieved fastest by the packet loss metric: While packet losses and linear link load start at almost the same level of weight changes, the decrease in weight change becomes less pronounced for the link load after a while (red) while the drastic drop continues for the packet losses (green), which results in a much faster convergence for the packet losses metric (dashed green line) than for the linear link load metric (dashed red line). In comparison, the nonlinear link load (blue) starts with fewer weight changes, but converges even later (dashed blue line). This is due to a slight beginning oscillation, which is most prominent in the weight plot (top row) and the bottleneck packet losses (4th plot from top). Increasing λ even further aggravates this oscillation tendency, while reducing λ to smaller values leads to significantly slower convergence times (plots not shown). The effect remains if we change the κ parameter of the nonlinear workload metric within reasonable bounds (plots not shown).

So, even though we initially assumed that the nonlinear packet loss metric looks like a good idea for capturing the fact that the network performance is not linear with the link load, we can conclude that this metric actually yields worse performance than the standard linear link load metric. The linear link load metric is comparable to the packet loss metric, but the latter yields a significantly faster convergence of the weight changes. By looking at it even closer, we see that the packet loss metric actually converges slightly faster than the linear link load in all plots apart from the overall drop rate (bottom plot).

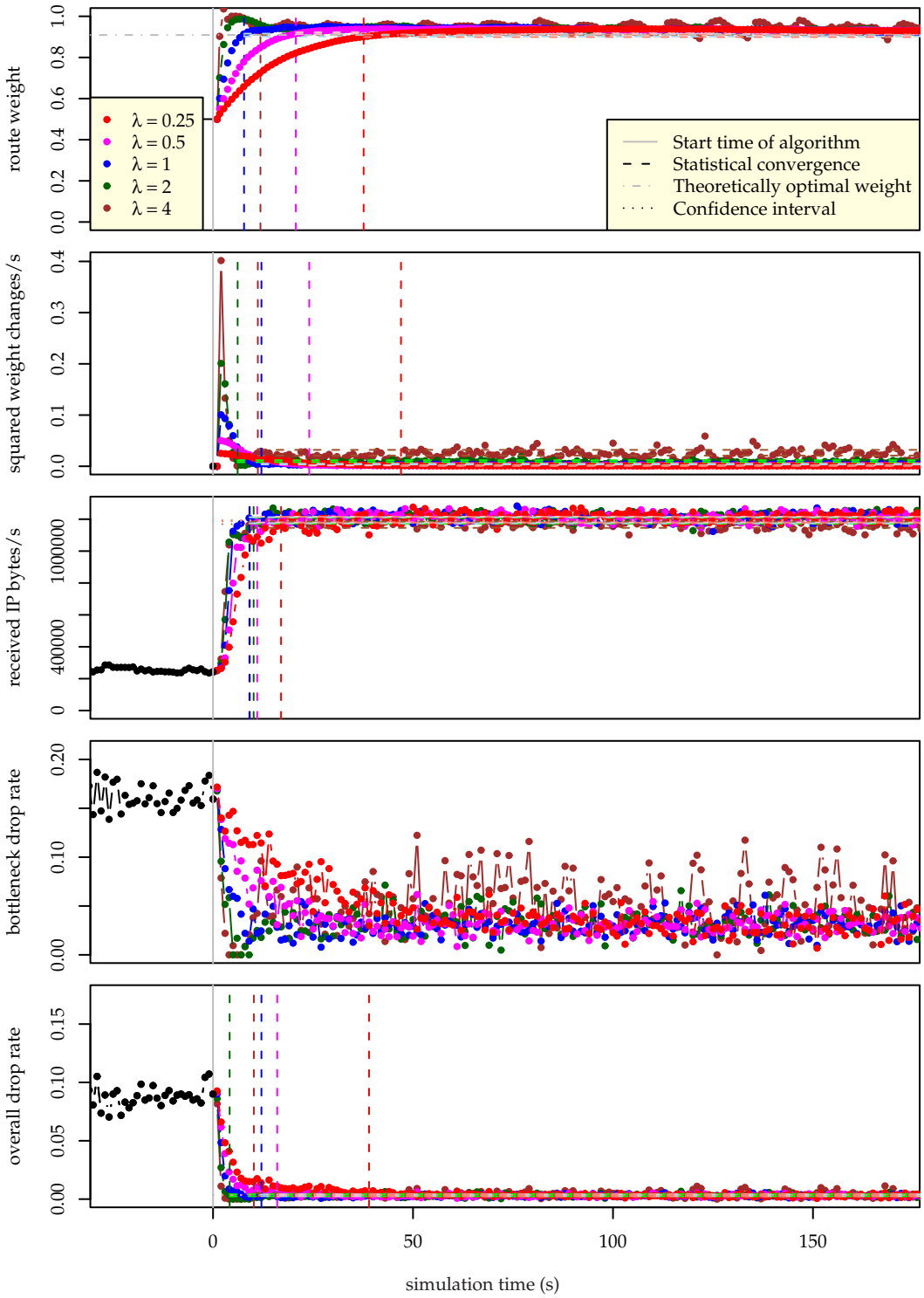


Figure 7.9: Packet losses as optimisation metric lead to faster convergence. In comparison to the link load metric, λ can be set to higher values without causing oscillation. Averages of 10 simulation runs.

7.3.4 Choosing a good λ for the packet loss metric

The packet loss metric thus seems to be the most promising optimisation metric, and is therefore the focus of rur further simulation experiments. Fig. 7.9 shows different choices for λ. We see that the metric begins to exhibit significant oscillations only at $\lambda = 4$. In fact, this effect starts to set in at around $\lambda = 3$ (plots not shown). As described earlier however, values of $\lambda > 1$ will lead to an initial swing-in phase (top-row plot, first 30 seconds). In order to rule out any negative influence that may be caused by this effect, we choose $\lambda = 1$ or the even more conservative $\lambda = 0.5$ in the remainder of our simulations, since both values still lead to a fast convergence (around half a minute) while not causing any oscillations.

7.3.5 The EMA parameter η

After having determined how fast RepleX should adjust its weight, we now attempt to find good parameters for the data source upon which it bases its weight change decisions: Remember that RepleX does not directly use its measurements and the values reported by the neighbours, but that it rather calculates an exponential moving average (*EMA*) on them, to remove the influence of ephemeral peaks and drops over a short timescale.

EMA

When calculating the EMA value for the current time interval, we weigh the newest measurement with a parameter η (with $0 < \eta < 1$) and the EMA value from the previous time interval with $(1 - \eta)$. Note that η has to be chosen wisely, since too large values *as well as* too small values potentially may lead to oscillations:

- If we choose η too large, we may risk that RepleX unnecessarily reacts to short-time traffic fluctuations; e. g., if we choose $\eta = 1$, RepleX does not do any averaging at all, but will rather operate upon the current measurement only. Since traffic fluctuations can be very pronounced (recall the bursty traffic shown in Fig. 7.2 on page 121), this easily can cause RepleX to overreact. Overreactions by one RepleX instance in turn provoke overreactions from others, i. e., through interactions with TCP and/or with other (overreacting) RepleX instances. These effects lead to oscillations.

- If we choose η too small, then our EMA-smoothed measurements will reflect changes in traffic patterns only very slowly, so that a RepleX instance does not get the feedback that it needs quickly enough. Rather, the RepleX instance can be lured into additional weight shifts away from some route that initially used to be congested, but that now is no longer congested. When the averaged measurements finally indicate that the congestion has been fixed, it is way too late, since by then the RepleX instance already has overreacted, i. e., shifted too much traffic. Naturally, after a while the same happens in the other direction. Especially in setups with several RepleX instances, their interactions can support the ensuing oscillations ad infinitum.

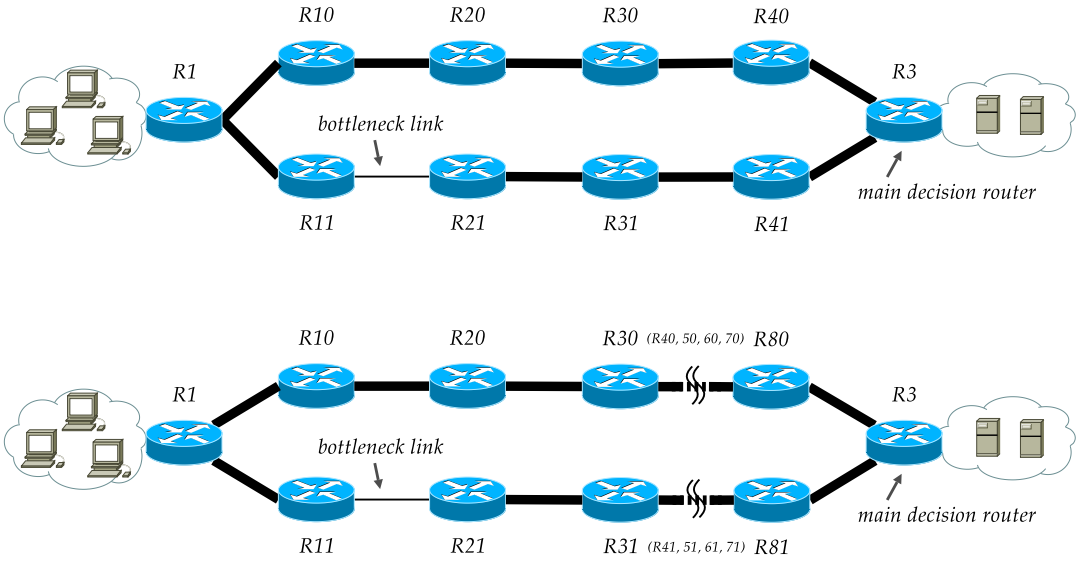


Figure 7.10: Topologies for testing the influence of η in relation to the number of hops.

Considering that not only the exponential moving average, but also the number of hops between a bottleneck and a deciding REPLEX instance influences the feedback speed, it thus makes sense to analyse the behaviour of our algorithm under different values of η, as well as with different hop lengths. In the simulations so far, we have used the topology in Fig. 7.5 on page 135, where the measurements at the bottleneck link $R21$–$R11$ cannot be made by the REPLEX instance that has to make the decision about shifting the traffic (i. e., $R3$). Rather, it has to be propagated one hop. To investigate other hop distances, we use the topologies shown in Fig. 7.10, and perform simulations with different values for η.

Some results of these simulation runs are shown in Fig. 7.11. Even from the first glance at the plots, it is evident that neither a drastic change to η (which used to be $\eta = 0.5$ in our previous simulations), nor the number of hops have a significant influence on the output of the algorithm. We get almost the same picture for simulations with larger hop distances (omitted for brevity).

In the top row plot we see that the initial weight changes are almost identical. As can be expected, convergence (dashed lines) is achieved faster for shorter hop counts and for larger values of η. The second plot confirms these findings for the weight change rate, albeit at later points in time. Note that the weight changes for all parameter sets start at the same time: We start our distance-vector like information distribution protocol (algorithm 4 on page 114; lines at bottom) *before* the REPLEX decision making process sets in at $t = 0$, so that the deciding REPLEX instance already has full information of the bottleneck route when it starts. This way, we exclude the influence of initial information delays and thus can compare the convergence times and oscillation behaviour based solely on the feedback delay.

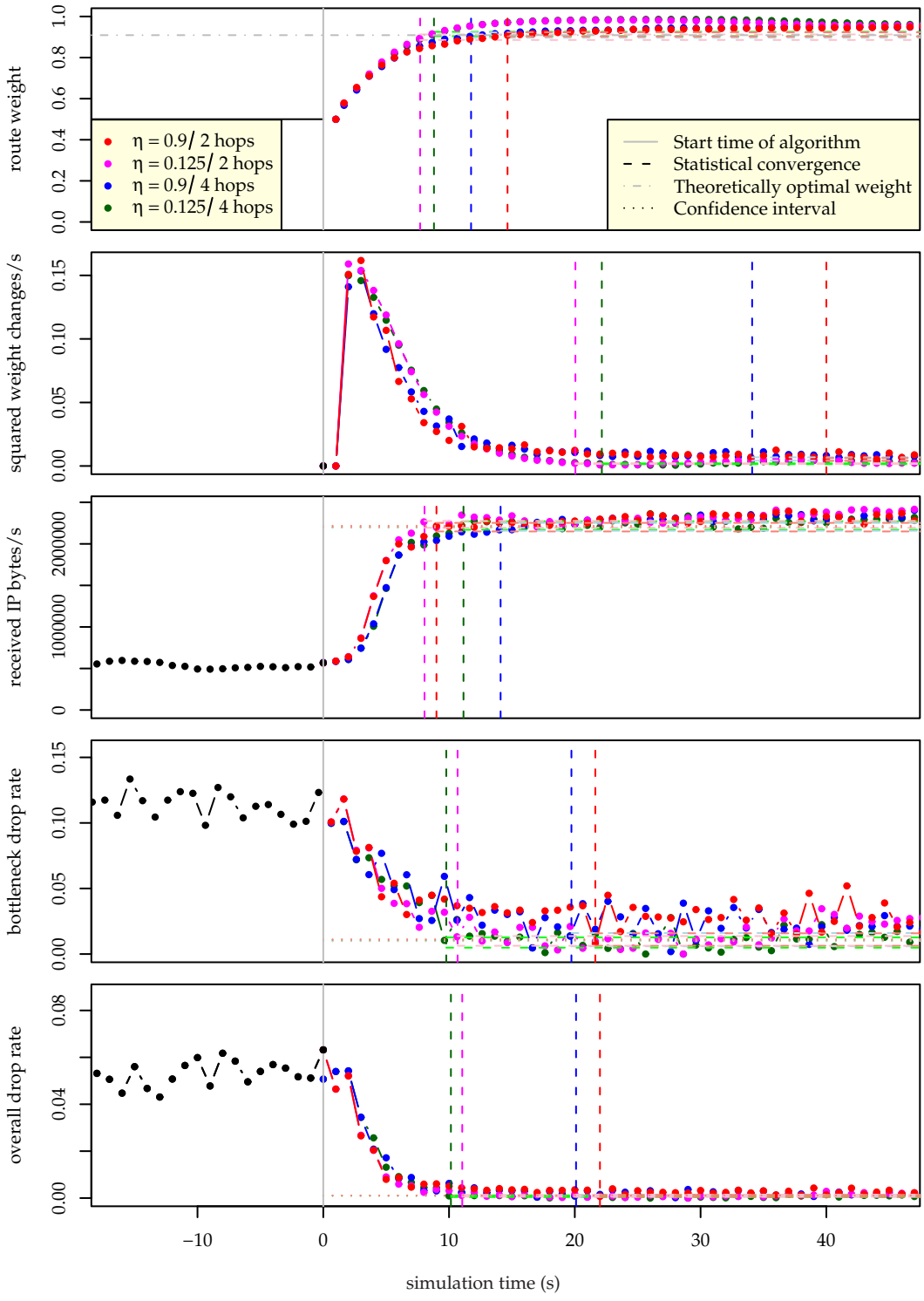


Figure 7.11: The EMA parameter η does not have a large influence on REPLEX's behaviour.

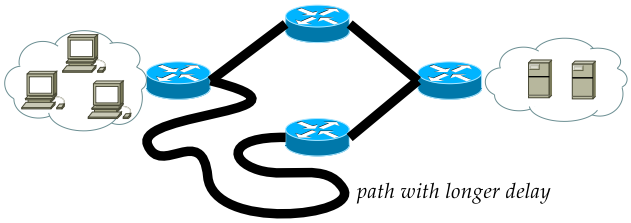


Figure 7.12: Topology for determining the REPLEX time parameter T.

The overall network performance (middle to bottom plot) improves on the same time scale as the weights, with the convergence being reached faster for the larger $\eta = 0.9$ and slower for $\eta = 0.125$, and is even less influenced by the number of hops.

In summary, we can conclude that for our settings with $\lambda = 0.5$, $T = 1\,\text{s}$ and packet drop metric, we can pick η from the wide range of values, independent of the diameter of our topology (i. e., maximum hop distance between any two REPLEX routers). We choose to stick with $\eta = 0.5$, since this in our opinion offers a good tradeoff between actuality and damping, as the current measurement contributes exactly one half to the EMA value.

7.3.6 Choosing the reaction time *T*

In our simulations so far, we have used an interval of $T = 1$ for REPLEX's main loop, which is much larger than the maximum RTT in our topology. This way, we ensure that the congestion control loops of actively transmitting TCP connections that were affected by a REPLEX-induced reroute have enough time to converge, before the next REPLEX decision may affect their congestion control loop again. Let us now investigate what happens if we make REPLEX react on shorter time scales.

REPLEX and TCP can interact in different aspects: First, an actively transmitting TCP connection may be redirected to a route that is more or less congested than the previous one; second, it may be redirected to a route with a larger or smaller network latency. The impact of these interactions on TCP's performance is especially bad if the network latency of the new route is much smaller than the previous latency, so that newer packets on the new route may overtake older packets on the old higher-latency route. Such *packet reordering* is treated by TCP as a packet loss, triggering the respective congestion control mechanisms. This development is especially contraproductive if the new route has the same or even a higher capacity than the previous route, as in that case obviously TCP should increase its sending rate, rather than decreasing it. Worse yet, if many TCP connections undergo these disadvantageous reordering events in parallel, they will enter their congestion control phase at about the same time, thereby *synchronising* and thus (due to their slow start algorithms being synchronised) creating unnecessary sudden packet bursts after some time, which makes the traffic even harder to handle.

packet reordering

synchronisation

Naturally, the smaller we choose our time parameter T, the more often this can occur. It is therefore interesting to investigate the time scale at which we can allow REPLEX to operate, without these negative effects getting overhand.

Another interesting aspect is the choice of the parameter λ in relation to T. Recalling Algorithm 5 on page 114, we see that the weight change is linear with λ and the difference in route performance. Assuming that the performance difference after a REPLEX decision does not change instantaneously but rather within a certain time interval, it thus seems rational to reduce λ if T is reduced, in order to avoid overreactions [FKF06].

To investigate the negative influence of smaller values of T (in combination with different settings of λ), we choose the following approach: We simulate a very simple topology that features two routes of equal capacity, but differing latencies; see Fig. 7.12. The upper path has a latency of 20 ms while packets on the lower path experience a delay of 70 ms. In this scenario, we cannot expect a significant improvement in network performance, if any at all, since the default 1:1 split ratio between the two routes of 10 Mbit/s each is the optimum already (unless our hashing method created a heavy imbalance in the actual traffic ratio through a random statistical fluctuation in client activity). However, since a re-route does dramatically affect the travelling time of packets through the network, we expect that running REPLEX on a time scale that is too small results in measurable performance decrease for TCP.

The latency difference between the two routes is 50 ms. In the extreme case that the 16,000 Bytes queue of one route is completely empty and the other one is completely filled, at 10 Mbit/s this difference can range between 37.2 ms and 62.8 ms due to the difference in queueing delays. While we conjecture that a change of at least 37.2 ms in delay for a route can be regarded as a "worst case" in an intradomain scope (satellite radio links aside), we chose this difference for another reason: With a maximum packet sizes of 1500 Bytes, this time difference is larger than the inter-packet time for any long-lasting TCP connection whose sending rate exceeds as little as 40 kBytes/s. In other words, any TCP connection that reaches a throughput of at least 3.2 % of the maximum link bandwidth of 10 Mbit/s is affected by REPLEX's reroute-induced packet reorderings.

In contrast to the previous simulations where we perform measurements only at the IP layer for performance reasons, we now also collect information on the number of bytes received by TCP (the *TCP goodput*), and the ratio of packet losses and reorderings detected by TCP. We take these metrics as an indicator for the effect of REPLEX on TCP performance. On the other hand, the performance indicators that we used to evaluate the previous simulations seem less effective for our purposes: The initial $1:1$ traffic split is already "converged", thus there is no temporal development to be expected, neither in the weight changes nor in the weights themselves; and the IP-level performance metrics (number of bytes, which also comprise REPLEX and OSPF traffic, but which do not explicitly reveal retransmits) are inferior to the direct TCP measurements. We therefore omit these figures for the following simulation runs.

TCP goodput

While values of $T > 1$ s can make sense in scenarios where network operators want to avoid frequent re-routes at all costs, we want to investigate the interactions between REPLEX and TCP on shorter timescales of 1 s and below. In previous work, we mentioned that the choice of λ and η actually depends on T, more precisely, on the ratios $\frac{\lambda}{T}$ and $\frac{\eta}{T}$ [FKF06]. Therefore, we run simulations for all combinations of $T = 1$ s,

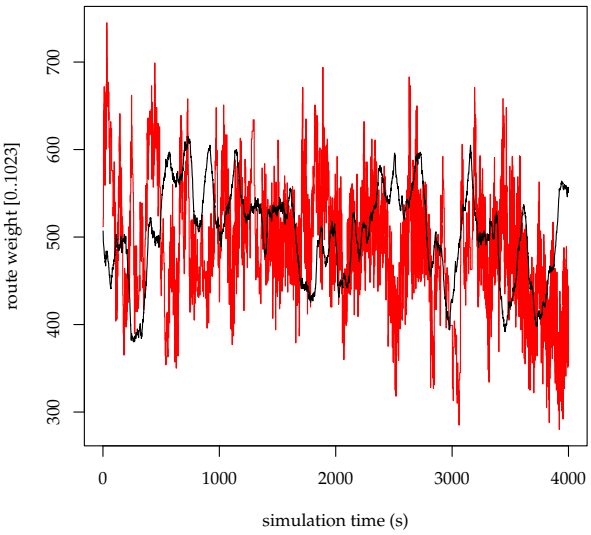


Figure 7.13: Example weight changes in a completely balanced setup. $\lambda = 0.5$, $\eta = 0.5$. Red: $T = 62.5\,\text{ms}$; black: $T = 1\,\text{s}$. One simulation run each with same random seeds for the workload.

$T = 0.25\,\text{s} = \frac{1}{4}\,\text{s}$, $T = 0.0625\,\text{s} = \frac{1}{4 \cdot 4}\,\text{s}$ and $\lambda = 0.5$, $\lambda = 0.125 = \frac{0.5}{4}$. We perform 10 simulation runs for each possible combination of parameters. As one can expect, these different settings lead to different behaviour of REPLEX in trying to adjust the weight settings; Fig. 7.13 gives an example for their difference in volatility.

To quantise the impact of REPLEX, we compare the values of TCP goodput and retransmission ratio. This allows us to relate the negative influence of REPLEX-induced reordering with the positive aspect of REPLEX-induced congestion reduction—note that we are analysing a scenario where traditional traffic engineering techniques cannot be applied, since the traffic is perfectly balanced over the two parallel routes of exactly the same capacity.

Table 7.1 shows the gain in TCP goodput (middle column) and the number of retransmits (right column) when we enable REPLEX with the specific parameters. Note that positive percentages are good for the TCP goodput, while they are bad for the number of retransmits (since more retransmits mean that the network is used in an inefficient way). Surprisingly and contrary to our initial expectation, the most volatile setting (small timescale of $T = 62.5\,\text{ms}$ and $\lambda = 0.5$) yields significantly better performance both in goodput (+17.84 %), as well as in the number of TCP retransmits (2.84 % reduction), whereas most of the more conservative, slower settings yield a performance that is slightly inferior to the scenario where we do not enable REPLEX. This is due to the fact that the highly volatile REPLEX settings can even react to statistical traffic spikes on a very short timescale, thereby significantly reducing the number of congestion-induced packet losses. This outweighs by far the number of retransmits that are caused

T (s)	λ	η	goodput	significant?	% retransmits	significant?
1	0.5	0.5	−6.14 %	✔	+0.13 %	✔
0.25	0.5	0.5	+2.89 %	✔	−0.23 %	✔
0.0625	0.5	0.5	+17.84 %	✔	−2.84 %	✔
1	0.125	0.5	−2.17 %	✔	+0.50 %	✔
0.25	0.125	0.5	−3.26 %	✔	+0.38 %	✔
0.0625	0.125	0.5	−2.99 %	✔	−1.09 %	✔

Table 7.1: TCP performance figures for different REPLEX reaction timescales. Averages of 10 simulation runs each. Statistical significance calculated by testing if confidence intervals overlap.

by packet reorderings. With a smaller λ however, REPLEX cannot react quickly enough. The same holds for time scales that are too large to react to the typical sub-second overload spikes, and for the case that the exponential moving average for the measurements changes too slowly (some simulations with $\eta < 0.5$; not shown).

To confirm that the performance differences are not caused by random fluctuations of our statistical workload, we calculate for each parameter set the mean TCP goodput confidence intervals (or the mean reordering percentage confidence intervals, respectively) over 10 simulation runs. We compare the confidence intervals that we obtain for simulations with REPLEX enabled against those that we obtain for simulations without REPLEX.[4] If the confidence intervals do not overlap, we take this as a strong indication that the performance difference really is an effect of REPLEX, and not of random fluctuations. The results of these tests are shown in Table 7.1 in the columns labelled "*significant?*". As we can see, all results are statistically significant under this test (marked by the tick symbol "✔").

We conclude that, while REPLEX potentially may cause packet reordering within TCP connections every time that it changes the route weights, this situation happens very rarely even under adverse circumstances. The effect of packet reordering is offset by the positive effect that REPLEX's potentially quick reactions to sudden spikes have on the network performance even in a perfectly traffic-engineered network.

As we believe that network operators want to rule out packet reordering as much as possible, we shall nevertheless continue our experiments with a setting of $T = 1\,\text{s}$, even though Table 7.1 shows that *this is not the most effective setting.* However, even this relatively conservative setting has shown in the previous experiments to let REPLEX converge rather quickly on a timescale of seconds or a few minutes, while not hurting the overall performance too much when there is nothing to be gained by traffic engineering. As another positive (and very relevant) aside, we save a significant amount

[4] The confidence intervals are calculated by dividing the simulation data into groups of 200 seconds of simulation time and taking their means. At this resolution, autocorrelation function plots (not shown) do not indicate that the groups are correlated with each other, which otherwise forbade us to calculate the confidence intervals. As the evaluation phase (or comparison phase, respectively) lasts 4000 s, we thus obtain $10 \cdot \frac{4000}{200} = 200$ non-correlated groups, for which we calculate the confidence interval for the mean of their means. [VR02]

of CPU time in our further simulations, especially in those involving more complex topologies with many REPLEX instances, since we do not have to increase the number of REPLEX calculations per second of simulation time.

7.3.7 Interacting REPLEX instances

The simulations that we have described so far comprise several REPLEX instances that communicate their performance measurements with each other. However, due to the network topology in use, all these simulations feature only a single REPLEX instance that make relevant decisions, which is on the router that sits at the fork of the two path alternatives near the server cloud. Only this instance has the full control over the traffic distribution in the network, whereas the purpose of the other REPLEX instances is solely to measure traffic and forward the relevant information to neighbouring REPLEX instances.

Our simulations so far thus only have demonstrated that *one* REPLEX instance does not fall into overreaction-induced oscillations, given a reasonable parameter set. But it remains to show that this holds if he have multiple REPLEX instances, each of which controls a fraction of the traffic. After all, REPLEX is in essence based on a *selfish* (i. e., non-cooperative) game, even though the actual REPLEX instances communicate and thus partially "cooperate" with each other, in order to make the necessary information for the individual decisions available for all.

The decisions of a REPLEX instance on some router R can interact with those of other REPLEX instances in three basic fashions:

- A REPLEX router upstream can decide to send more (or less) traffic to R. This may lead to increased (or reduced) congestion on some of R's outgoing routes, which in turn requires R to take action in order to offset the changed situation.
- The same situation also can be seen from the other end: If R decides to change some route weight, this automatically changes the performance figures for the affected routes. This change is then reported to other routers upstream, which in turn may increase (or reduce) the now more (or less) performant route via R.
- Another interaction situation arises when two or more REPLEX instances compete over a bottleneck link further downstream. Recalling the experiences in the Arpanet [KZ89], one may intuitively expect a high potential of mutual overreactions and thus oscillations for such a scenario.

To analyse all three of these interactions between REPLEX instances, we simulate the network shown in Fig. 7.14, with 350 clients at $R1$ and 40 servers at $R3$. As usual, the majority of the traffic is flowing from the servers to the clients; so we can again neglect the traffic going in the other direction. Let us now consider the structure of this topology.

As the traffic enters $R3$, it encounters first bottleneck. The measurements performed by $R3$ will suggest a traffic split in a $2:1$ ratio for $R20$ vs. $R21$. However, if we look

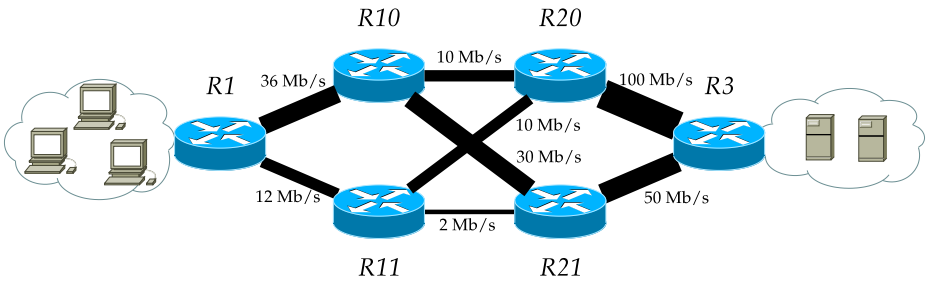


Figure 7.14: Topology for analysing interactions of multiple REPLEX instances.

further downstream, we see that this is not the real bottleneck, which actually sits between the routers $\{R20, R21\}$ and the routers $\{R10, R11\}$. Considering the link capacities downstream, it is obvious that $R3$ should split its traffic not $2:1$, but rather roughly $5:8$—i. e., actually the other way around.

There is, however, another important aspect to note: As soon as REPLEX starts, the route quality as reported by $R21$ will be vastly inferior to the one reported by $R20$—while $R21$ has, in theory, the greater capacity, it initially exerts a downstream traffic split of $1:1$ towards $R10$ and $R11$. Considering its input capacity, this results in most of the capacity of link $R21 \rightarrow R10$ being unused, while imposing an enormous overload on the link $R21 \rightarrow R11$ at the same time. This will greatly deteriorate the overall performance that $R21$ reports back to $R3$. On the other hand, the downstream links of $R20$ are balanced; therefore, $R20$ will not suffer from artificially bad performance. Hence, $R3$ initially is likely to shift traffic away from $R21$, rather than shifting towards it. Also note that $R20$ and $R21$ compete over the link $R10 \rightarrow R1$: The capacity of this link (36 Mbit/s) is slightly smaller than the sum of the inbound capacities at $R10$ (40 Mbit/s), making it a further bottleneck. This situation is aggravated by the fact that the links $R20 \rightarrow R10$ and $R21 \rightarrow R10$ have relatively high capacities and thus are attractive targets for the REPLEX instances at $R20$ and $R21$. The parallel link $R11 \rightarrow R1$, in turn, is not a bottleneck. Thus, while $R20$ and $R21$ "fight" over $R10 \rightarrow R1$, this will indirectly influence the decisions of $R3$ at the other end, regarding its traffic split between $R20$ and $R21$.

To summarise, in spite of its relatively small size, this topology is full of potentially adverse interactions between the different REPLEX instances. It is thus ideally suited for exemplarily studying the behaviour of our algorithm in a competitive scenario. At the same time, it is small enough to allow us to study it in closer detail.

Results of this simulation are shown in Fig. 7.15 on the next page. We see that the weights converge very quickly (top row plot), with the exception of the weights at $R3$ (blue line): At start, $R3$ very quickly changes the weights, but then overshoots to a setting of about 80 % to $R21$, which afterwards slowly is taken back a bit during a time range of several hundred seconds. $R21$ very quickly finds out that directing almost all traffic towards $R10$ and thus avoiding the extremely under-dimensioned link to $R11$ is a good idea, and during the next ≈ 3000 s very slowly increases this value even further.

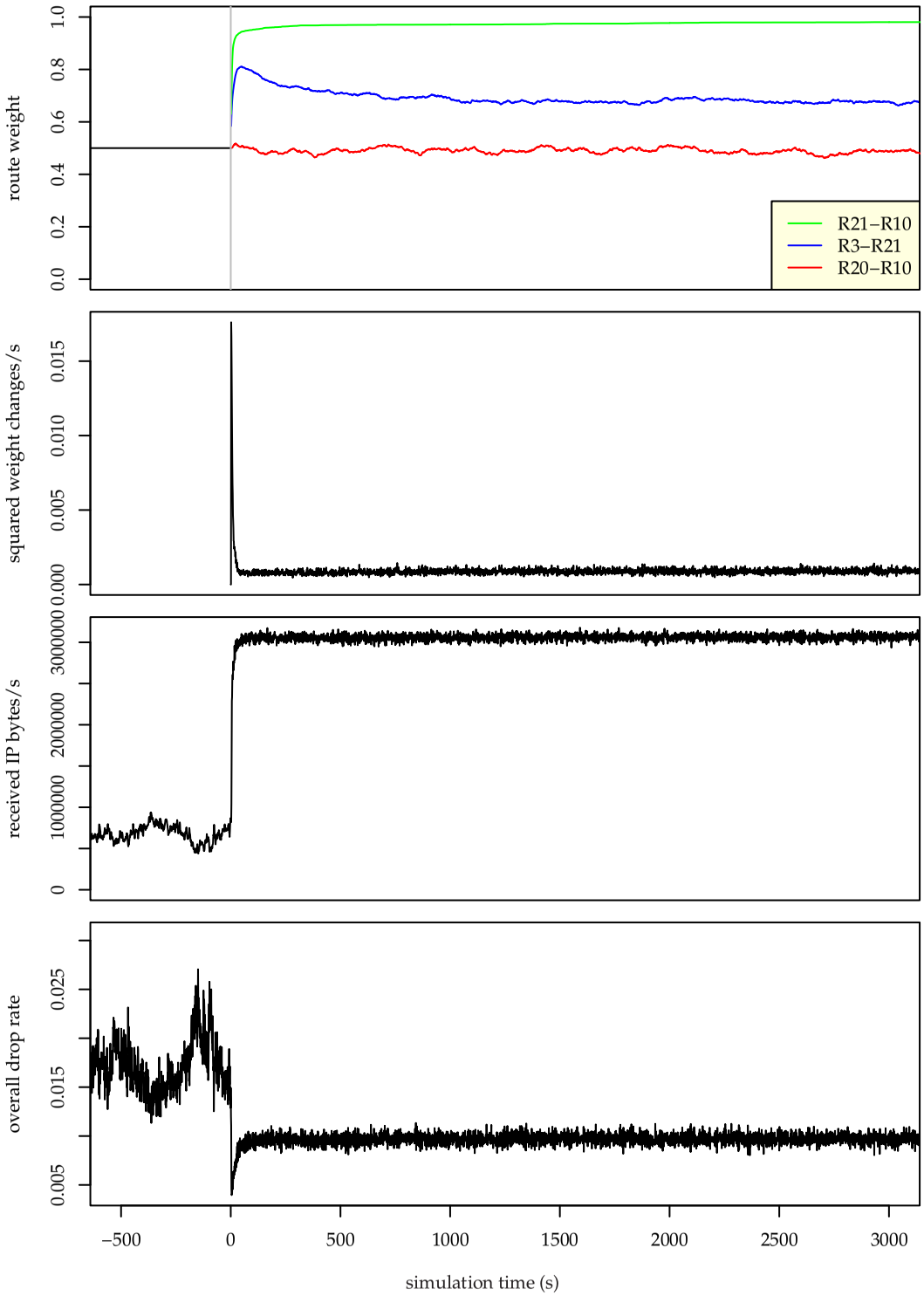


Figure 7.15: Results for interacting REPLEX instances.

Neither of these longer-term developments manifest themselves in the absolute volatility of REPLEX (second plot from top): The most drastic weight changes take place within the first < 30 s (sharp spike), after which obviously limited changes occur, which are in majority caused by statistical workload fluctuations on a small timescale.

The network performance metrics evolve in the same manner: A drastic increase in throughput is gained within the first few seconds (plot in 3rd row), after which the situation does not measurably improve any further. Note that, just as in previous simulations, the traffic becomes significantly less volatile as soon as REPLEX is activated: Without REPLEX being active, some long-running TCP connections find themselves to be "fortunately" (i. e., through pseudorandom hashing) assigned to uncongested high-capacity routes, while most are directed to congested routes.

The overall drop rate $\frac{\sum \text{drops on any path towards R1}}{\sum \text{packets sent from R3 to R1}}$ (bottom row plot) evolves more or less in the same way (with a different sign), with one remarkable exception: Right after the activation of REPLEX, the overall drop rate almost reaches 0 and then slowly increases again to a new, but lower and less volatile level.

When we look at the developments more closely, i. e., with a higher temporal resolution, we see the explanation: At the very start, $R21$ shifts a lot of traffic (Fig. 7.16 on the next page, top row, green line) away from the heavily congested link to $R11$ and towards the uncongested link to $R10$. This immediately manifests itself in a very sudden drop in packet losses on this bottleneck link (bottom plot; straight light green line), which previously featured a dramatic packet loss rate of almost 30 %, i. e., prior to the start of REPLEX.

Note that, even though $R21$ shifts most of its traffic towards $R10$, this link features almost no packet losses (Fig. 7.16, bottom plot, dashed dark green line at $y = 0.00$). The reason why $R21$ does not put even more traffic on this link is that the drop rate at $R10 \rightarrow R1$ is slightly above zero, since this link is slightly underprovisioned (Fig. 7.14).

Another interesting development is that the drop rate at $R3 \rightarrow R21$ increases after a while. This is due to the fact that REPLEX immediately shifts more weight to $R21$ (top row plot, blue line shows the inverse $R3 \rightarrow R21$), and is further aggravated by the fact that the REPLEX instance at $R21$ removes the downstream bottleneck $R21 \rightarrow R11$, so that now the bottleneck property of $R3 \rightarrow R21$ becomes more pronounced. Since we use TCP traffic as workload, this added network performance along the route will be quickly detected by active TCP connections, which almost immediately start sending packets in ever shorter intervals, thereby quickly reaching the capacity limit of the new bottleneck link $R3 \rightarrow R21$. In other words, this development is a result of a positive interaction between TCP and REPLEX.

We see another notable interaction at $R20$: Actually, the route $R3 \rightarrow R20$ is assigned *less* weight (top-row plot, blue line), but at the same time, the drop rates on the outgoing links of $R20$ start to *increase* (3rd plot from top, dashed red and solid orange line). How can that be? Here, the answer is not just TCP, but rather our workload model: As we on purpose disabled persistent HTTP connections, each HTTP object is transferred in its own separate TCP connection. During the download of one page, many objects have to be transferred (e. g., the HTML page itself, some inline images, style

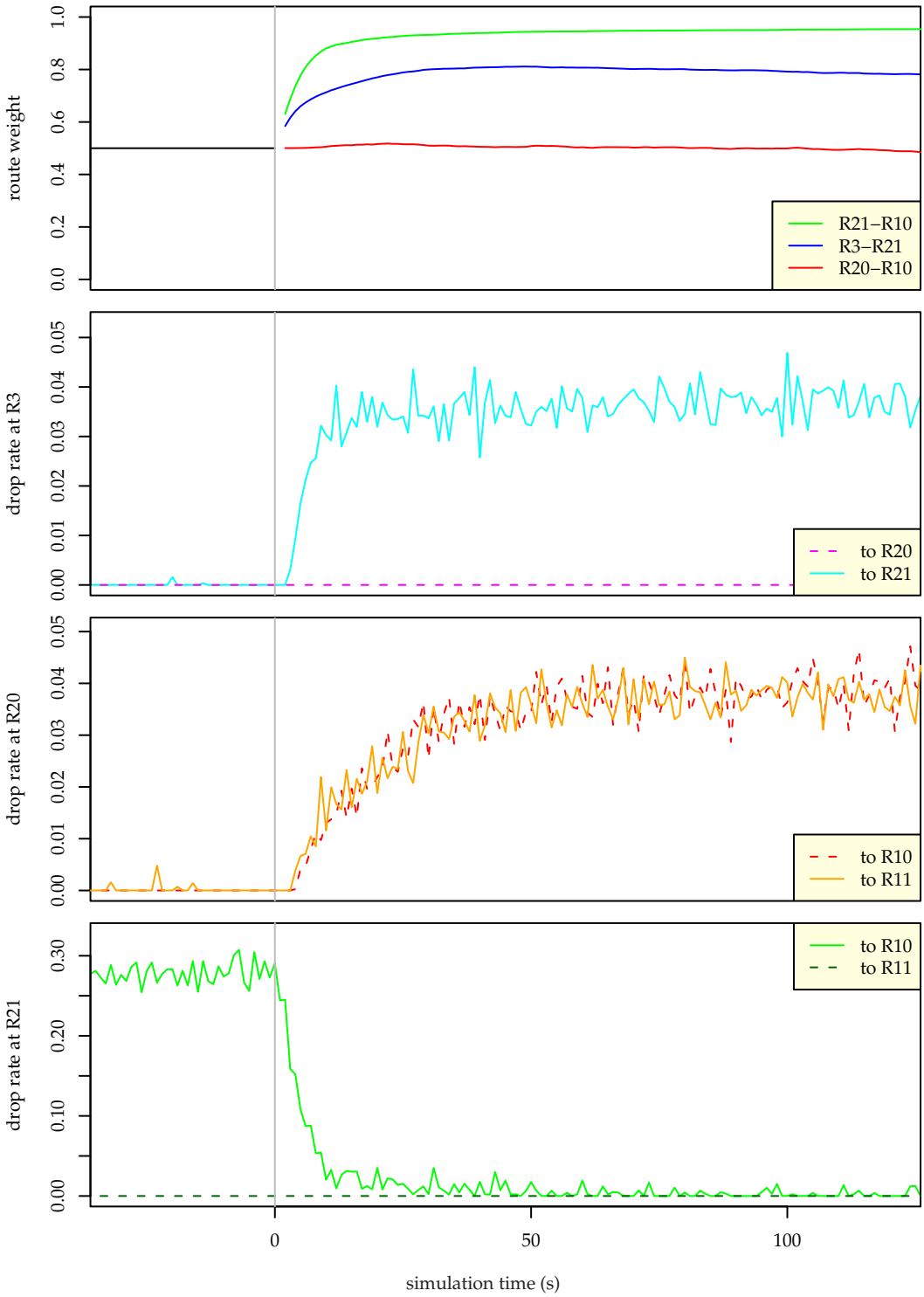


Figure 7.16: Results for interacting REPLEX instances (detail).

sheet files, Flash objects, etc.), thus creating many rather short-lived TCP connections in a short interval of simulation time. However, these downloads do not all take place at the same time; rather, a client downloads just two objects in parallel, and only opens a new TCP connection for a new download when a previous download has finished. Moreover, we have a similar setting for the peer-to-peer workload. Now, if the TCP performance on the *other* routes (i. e., not via $R20$) is drastically increased, this means that the HTTP transfers via these routes finish in a shorter time, thereby allowing the HTTP client to start its next download somewhat earlier. This hence leads to an increased download rate, and thus in a higher rate of new TCP connections that are "randomly" (i. e., through hashing) assigned to a route via $R20$. With this added traffic, $R20$ now becomes a bottleneck. Since this development is caused not only by an interaction of REPLEX and TCP, but rather between REPLEX, TCP, *and* the workload model, this development is slower than on the other links, and takes place on a timescale of about one minute (3rd plot from top, dashed red and solid orange line).

In summary, we can see that REPLEX yields an improved network performance even if the decisions of different REPLEX instances influence the decisions of other instances, and even interact with the simulated workload-generating "users" at the TCP end hosts. Oscillation does not occur, and a converged state is reached very quickly.

7.4 Realistic topologies

So far, we have analysed REPLEX's behaviour in a number simulations featuring relatively simple topologies. While the purpose with these simulations is to find out good values for the various REPLEX parameters (mainly λ, η, T, and the metric to optimise), we in this section are now going one step further by analysing the performance of REPLEX in simulations involving realistic topologies. In addition, we compare the performance of REPLEX to that of traditional traffic engineering by tuning OSPF link weights, and check furthermore the behaviour of REPLEX with communication between the routers being disabled.

We focus our analyses on the EBONE (AS 1755), Exodus (AS 3967), Metromedia Fiber / Abovenet (AS 6461) network topologies from the 2002 Rocketfuel [SMW02] data, and the AT&T topology which is shipped as an example together with the TOTEM toolbox [BLM]. We chose these topologies, as they feature a reasonable number of routers and links. Moreover, their good connectivity offers good path diversity, which our algorithm can take advantage of.

7.4.1 Size reduction

Since the Rocketfuel maps are heuristically obtained from active measurements, they are not necessarily 100 % accurate. Moreover, they are lacking some important information, such as link bandwidths and, in many cases, link latencies, which we therefore have to estimate. Since Rocketfuel topologies contain the (estimated) geographical location of each router, we can calculate the minimum physical length for each link based on the speed of light in glass fibre [Pop, Hen].

A full Rocketfuel map yields a rather complex network topology comprising many routers, typically more than one hundred. While we did indeed perform several simulations of such large networks with realistic workload generators [FKF06], CPU and memory consumption of these simulations proved to be too time-consuming to allow many simulations that span a sufficiently long simulation time. We therefore decided to reduce the Rocketfuel maps in size while retaining their characteristic structure.

We note that many of the Rocketfuel maps already are available in reduced forms ("`r0`", "`r1`" maps, etc.). In their case, the reduction has been performed by identifying routers that are located near the edge of the network and removing them. While this leads to significant reductions in network size, artefacts of this reduction strategy can lead for some parts of the network to disappear completely, while other parts retain unnecessary detail. For example, we need to set reasonable link delays; yet Rocketfuel link delay estimates cover only a small number of links. Our strategy for estimating missing link delays is based on geographical distance. But this only works for those links that connect different cities; it is not possible for us to estimate link delays within a single PoP this way. However, some of the reduced maps retain the internal structure of PoPs, while possibly erasing other PoPs altogether. Removing entire PoPs, on the other hand, is something that we want to avoid, since links between PoPs normally are long-distance links, i. e., an expensive resource, whose utility we want to maximise through the application of REPLEX. Thus the Rocketfuel standard reduction technique provides us on the one hand with detail which we cannot use, but on the other hand removes important parts which we would like to retain. Therefore, the "`r0`" and "`r1`" maps are not suitable for our purposes.

We therefore implement a different reduction technique based on geographical positions. For (almost) every router, the Rocketfuel maps provide us with the name of the city where that router is (supposedly) located. In a preparatory step, we remove all routers from the topology with unknown location, if any such routers exist. In our main reduction step, we then aggregate all routers that are located in the same city into one "meta"-router. Our reduction technique thus yields a topology that retains those links that are most cost-intensive—i. e., the long-distance links that connect different cities—while removing unnecessary detailed descriptions of the structure of single PoPs. The size reductions are significant. Table 7.2 compares the sizes of original topologies and our reduced versions. Routers marked in the Rocketfuel topology as being external are ignored. With our reduction technique, we can typically achieve a reduction factor of at least 1:10, while retaining the geographical distribution of the network.

7.4.2 Traffic demands and link parameters

In the previous simulations, we maintained a cloud of workload clients on one side and a cloud of workload servers on the other side of the topology, in order to obtain a main traffic flow, which makes the analysis easier. In our further simulations however, do not any more distinguish between client and server workload clouds. Rather, we install workload clouds containing some number of clients and some number of servers at every router of the simulated topology.

AS #	RF routers	RF links	red. routers	red. links
1755	299	548	27	46
3967	446	920	21	35
6461	654	1338	22	54
(AT&T)	(154)	(188)	—	—

Table 7.2: Comparison of original and reduced Rocketfuel topologies (*RF*=Rocketfuel, *red.*=reduced). We let the AT&T topology from the Totem project in unreduced form.

In all of our simulations, the request distribution between clients and servers is uniform, and therefore the effective traffic demands between each client/server pair are uniformly distributed—i. e., at all times, each client chooses the Web server hosting the next Web page to be downloaded uniformly, regardless of their location. The number of servers in a workload cloud thus determines the amount of traffic flowing from the location, while the number of clients determines the amount of traffic flowing towards it. In other words, the number of clients and servers in each workload cloud directly determines the traffic matrix.

Since we do not know the traffic matrix for the given topologies, we choose to install the most basic setup: We set the number of clients and servers to be the same within every workload cloud. Hence, we achieve a uniform traffic matrix.

Rocketfuel estimates only the network topology and link latencies, but not link bandwidths. We use the former in our simulations while the latter have to be chosen manually. As traffic engineering is most effective in a network where the bottleneck is in the backbone and not at the periphery, we simulate an *underprovisioning* network and set the link bandwidths of the links between routers to 30 Mbit/s while ensuring a saturation of many links through an appropriate number of clients, ranging from 20 to 40 clients and 7 to 10 servers per router, depending on the topology.

7.4.3 REPLEX setup

As in the previous simulations, we choose $\lambda = 0.5$, $\eta = 0.5$, $T = 1\,\text{s}$, *metric* = drop rate, $\alpha = 0.25$, $\beta = 0.1$, $\epsilon = 0.0$ as parameters for REPLEX. The time interval implies that one pair of REPLEX messages is exchanged per second between every pair of neighbouring routers. Assuming that each REPLEX message contains a 6-byte information for 2,000 destination prefixes, this amounts to a total of just 96 kbit/s, i. e., just 0.32 % of additional traffic on each 30 Mbit/s link.

Note that, due to the high memory demand (3.5–7 GBytes) and the long running time (one day up to a week), we do not provide multiple simulation runs for the same simulation to reduce the influence of statistical fluctuations. Rather, we simulate each parameter set only once.

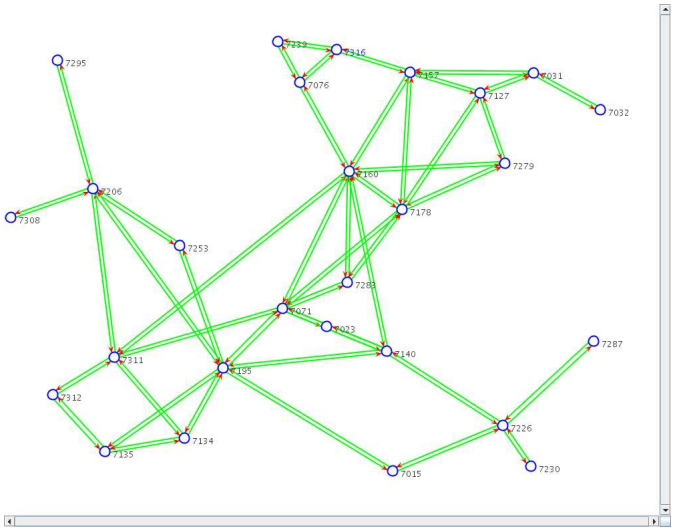


Figure 7.17: Reduced Rocketfuel topology for AS 1755.

7.4.4 Effectivity of REPLEX

We start with analysing the weight change behaviour of our algorithm in the topology AS 1755. The topology is shown in Fig. 7.17. It consists of 46 links connecting 27 routers, each running a REPLEX instance. All routers are connected to workload clouds and thus are workload traffic destinations. It is therefore not feasible to show weight changes for individual links; neither does it make much sense to show weight changes for specific routes at specific nodes. Instead, we consider the total weight changes for each 1 second interval. As with the previous experiments in Section 7.3, we expect the weights to be subject to small fluctuations even in the converged state, due to the bursty nature of the generated realistic workload traffic.

Fig. 7.18 shows the temporal development for weight change metric (i. e., like the plots in the second row in Fig. 7.6 ff). We see a sharp decline during the first $\approx 150\,\mathrm{s}$. A converged state by our definition from Section 7.2.2 is reached roughly at $t = 400\,\mathrm{s}$ (dashed vertical line).

While convergence certainly is a pleasant property, it remains to show that REPLEX actually improves network performance. To demonstrate the performance gains, Fig. 7.19, shows the overall network throughput, i. e., the total number of IP bytes that are received by all simulated clients per 1 second time interval. We see that the network performance increases slightly faster than the weight changes decrease (within a time window of roughly 200 s). Note that, in contrast to the weight changes, it does not make sense to apply our convergence definition, since here the range of our measurements (i. e., throughput values) prior to and after starting REPLEX do overlap. Thus, we cannot be sure that reaching the convergence corridor is a sign of convergence, or just an outlier caused by a random fluctuation in the workload that might as well have happened without REPLEX. However, we can see that the throughput gain achieved by

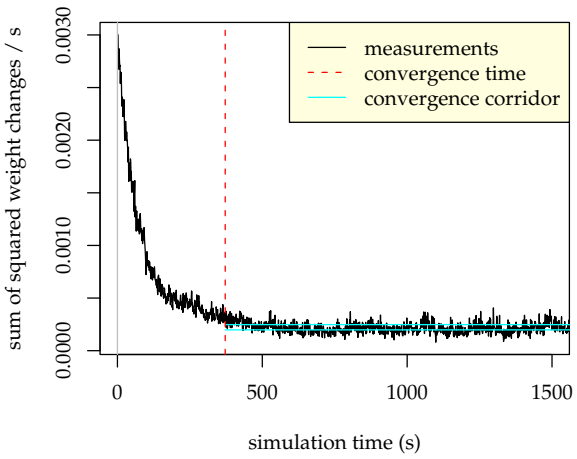


Figure 7.18: Weight changes in AS 1755.

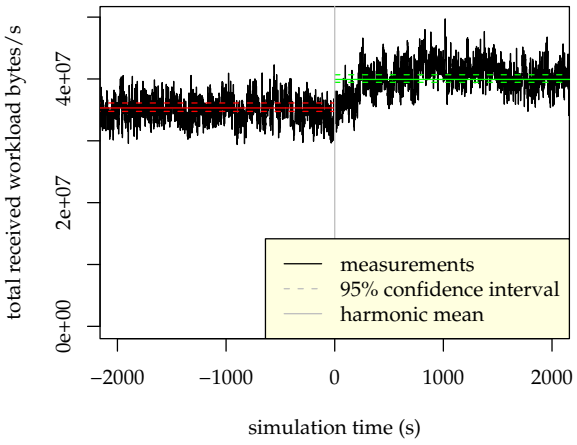


Figure 7.19: Influence of REPLEX on network throughput in AS 1755.

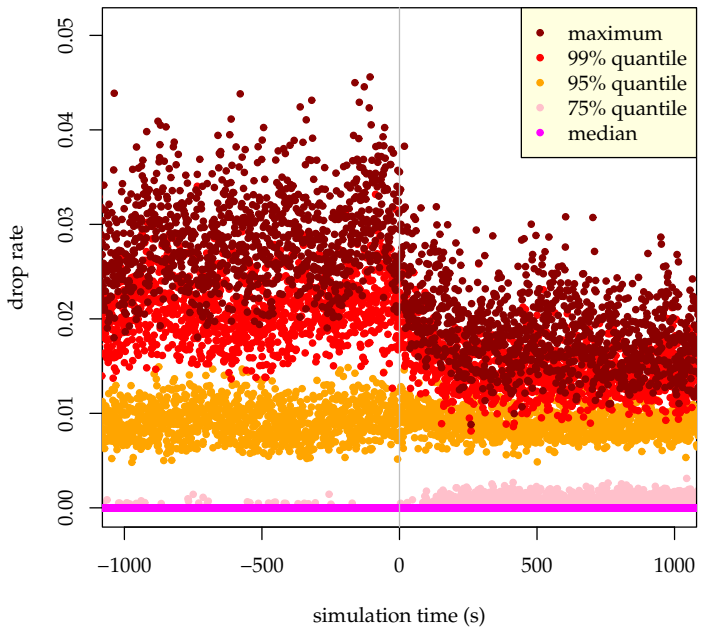


Figure 7.20: Influence of REPLEX on the drop rates in AS 1755.

REPLEX is significant, since the 95 % confidence intervals[5] are clearly separated (dashed red and green lines).

Where do these gains in throughput come from? Naturally, the reason is a significant reduction in the number of packet drops. Fig. 7.20 shows the maximum drop rate of all drop rates in the network (brown), as well as their 99 % quantile (red), the 95 % quantile (orange), the 75 % quantile (pink) and the 50 % quantile (i. e., median, flat magenta-coloured line at bottom). We see a swift decrease in the maximum number and 99 % quantile of packet drops during the first $\approx$ 250 s after REPLEX has been started. The 95 % quantile remains more or less unchanged, while the 75 % quantile slightly increases. The median of the link loads (and thus all lower quantiles as well) remains constantly at zero. In other words, a strong reduction of the most heavily congested links (brown, red) is achieved, at the cost of a slight increase in the loss rate on less congested links (pink).

It is interesting to correlate the temporal development of the drop rate with the link loads. Fig. 7.21 reveals that the most heavily loaded links, which naturally are those suffering from the highest drop rates, are only slightly relieved in terms of link load (brown and red points)—but remember that this small load decrease by scarcely 10 %

[5]This and the following confidence intervals are calculated from 32 group means of 125 measurements each. ACF plots (not shown) gave no indication for autocorrelation-induced nonstationarity.

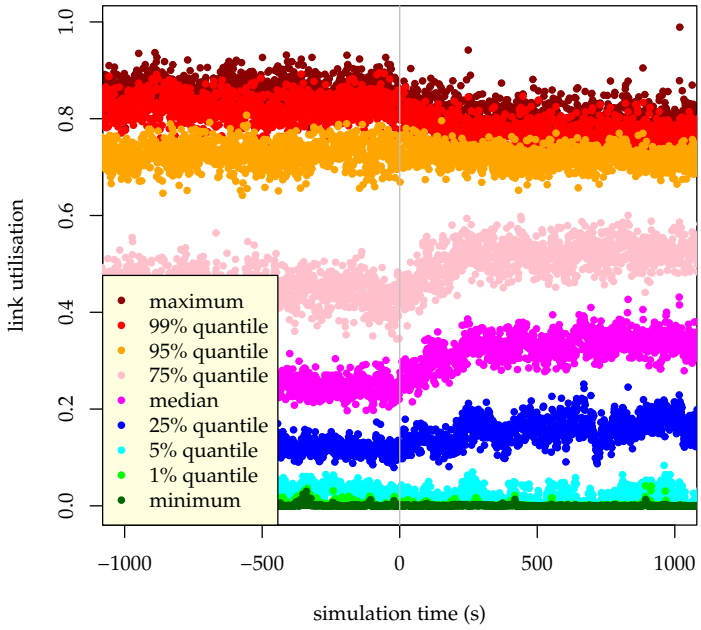


Figure 7.21: Influence of REPLEX on the link loads in AS 1755.

corresponds to a packet drop decrease of 30–40 % (recall Fig. 7.20).[6] Apart from a few links that carry almost no workload (5 % quantile), the load on the majority of the other links is significantly increased (25 % to 75 % quantile). In particular, the 75 % link load quantile (light pink) is increased from about 50 % to almost 60 %, which corresponds to an increase in packet losses from virtually zero to about 0.2 % (Fig. 7.20, pink)—which is consistent with the common practice among network operators of not letting the load on any link exceed 50–70 %, since otherwise performance starts to degrade.

We see a similar behaviour in our simulation of AS 3967 (topology shown in Fig. 7.22 on the next page) instead of AS 1755: Most links, even the very lightly loaded ones, experience a significant increase in utilisation after we enable REPLEX (Fig. 7.24 on page 163). In contrast to AS 1755, the most heavily loaded links do not experience much of a visible decrease in link load. However, these most heavily congested links are nevertheless relieved in terms of packet losses (Fig. 7.23 on the next page), albeit less pronounced than in the case of AS 1755. All in all, this development also leads to a significant increase in network throughput (Fig. 7.29).

The results of these and other simulations are summarised in Table 7.3 on page 164. We witness a performance increase of 13.1 % for REPLEX over the unoptimised network

[6] When regarding these very high values for maximum loss rates of around 3 % and the maximum link loads of about 90,%, keep in mind that this is on purpose, since we are simulating a vastly *underprovisioning* network.

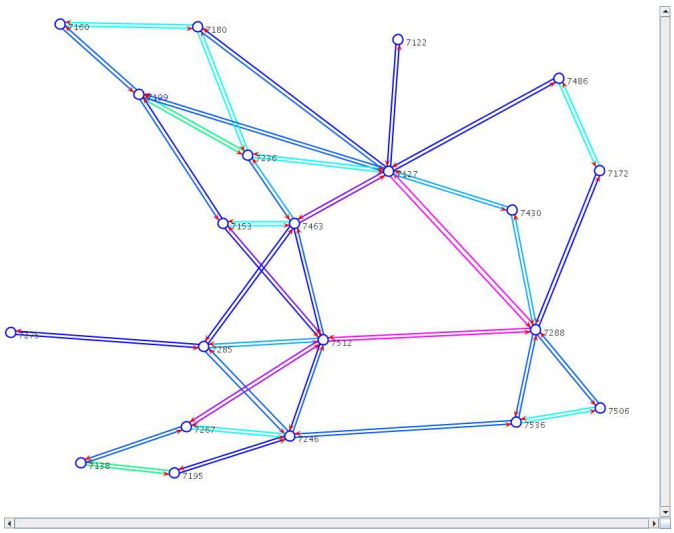


Figure 7.22: Reduced Rocketfuel topology for AS 3967.

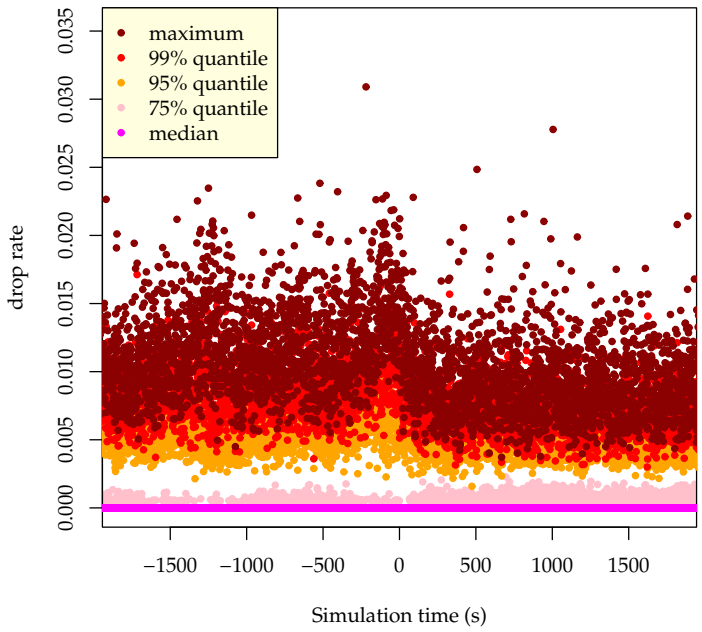


Figure 7.23: Influence of REPLEX on the packet losses in AS 3967.

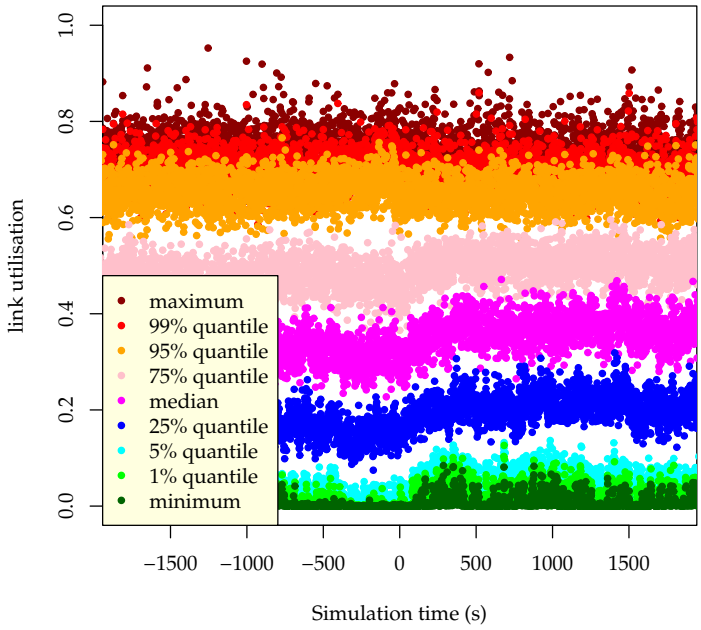


Figure 7.24: Influence of REPLEX on the link loads in AS 3967.

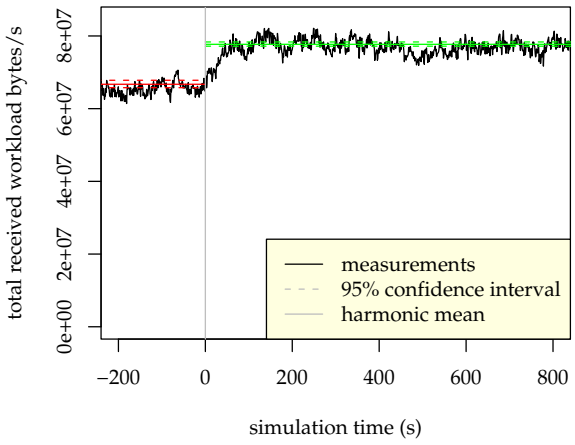


Figure 7.25: Influence of REPLEX on the throughput in the AT&T topology.

Topology	Optimisation method	TP gain over unopt.	signif.?
1755	IGPWO	+10.7 %	✔
1755	REPLEX	+13.1 %	✔
1755	REPLEX + IGPWO	+11.2 %	✔
1755	REPLEX/noComm	+5.4 %	✔
3967	IGPWO	+6.5 %	✔
3967	REPLEX	+8.5 %	✔
3967	REPLEX + IGPWO	+7.6 %	✔
3967	REPLEX/noComm	+7.1 %	✔
3967	REPLEX/noComm + IGPWO	+7.0 %	✔
6461	REPLEX	+6.5 %	✔
AT&T	REPLEX	+16.4 %	✔

Table 7.3: IP throughput (TP) gains over the unoptimised network (unopt) for REPLEX, REPLEX without communication (REPLEX/noComm), traditional IGP weight optimisation (IGPWO), and combinations of these.

in the 1755 topology (first block, second row), and of 8.5 % for the 3967 topology (second block, second row). When simulating AS 6461, we also notice a statistically significant performance increase of 6.5 % through the use of REPLEX (third block). Furthermore, we achieve a very good performance increase of 16.4 % for the AT&T topology (fourth block). Here, the network throughput is less volatile (Fig. 7.25 on page 163), and the new performance level is reached within less than 100 seconds. This quick performance increase is again correlated with a quick drop in weight changes. When we examine the weight changes of all routes at one particular router (Fig. 7.26 on page 165), we see that indeed many weights converge within less than 100 s (left). In many cases, we observe weight settings where one route alternative is completely disabled (i. e., either weight $= 0$ or weight $= 1$). Other route weights, in contrast, do not change much, if at all (horizontal and almost horizontal lines starting at $y = \frac{1}{2}, y = \frac{1}{3}, y = \frac{1}{4}$). Yet other routes show a slow convergence towards some value, e. g., the two dashed red lines that converge from 0.5 to around 0.7. Even other routes keep changing around $y = 0.5$. Another effect that can be seen is that many routes share the same bottleneck link and therefore almost behave identically. This results in weight change lines that are almost parallel.

In summary, we have seen that REPLEX also converges quickly and without oscillations in realistic topologies, while at the same time resulting in an improved network performance. [7]

[7] Note that the remaining rows of Table 7.3 are addressed in the following sections.

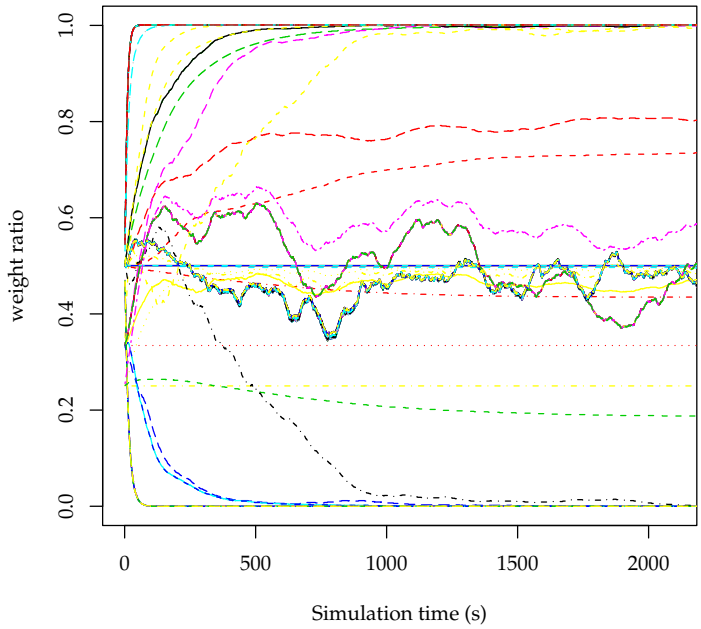


Figure 7.26: Weight changes at one router in the AT&T topology.

7.4.5 Comparison to traditional IGP weight optimisation

So far, we have shown that REPLEX converges fast, that it does not lead to oscillations if sane values are chosen as parameters, and that it yields an increased network performance. However, another important question is: How well does REPLEX compare against other traffic engineering methods?

To demonstrate its potential, we decided to compare REPLEX against a very popular TE method: the traditional technique of adjusting OSPF (or other IGP) link weights [FT00]; refer to Section 2.3.3 on page 27 for further details. The TOTEM traffic engineering toolbox [BLM] provides an implementation of this method. In the TOTEM toolkit, it is called *IGPWO* (for "IGP weight optimisation"), and we adopt this terminology in the remainder of this document.

IGPWO

Recall that IGPWO requires the full traffic matrix as input. The quality of its output is, of course, dependent on its input. In many cases, network operators do not employ Cisco NetFlow or comparable techniques to determine a (nearly accurate) traffic matrix due to performance reasons; they rely on traffic matrix estimation techniques instead. However, even if they do not estimate but have access to accurately measured TMs, it has to be noted that any traffic matrix obtained this way can only reflect the traffic demands that are possible within the capacity of the current network setup—these measurements can, however, not anticipate any change in traffic demand that is caused

through a routing optimisation (i. e., additional traffic demand that can now be satisfied due to an increased network performance along a previously congested path). In other words, even if the true traffic demands remain constant, the measured traffic matrix will not, and is thus always an imperfect input to the IGPWO algorithm, even if the actual measurements perfectly capture the currently imposed traffic demands. In our case, we actually do *know* the traffic matrix with the *true* traffic demands—because we have set up our simulation in a way such that it is uniform. This means that we can give the IGPWO algorithm a perfect input from which it can estimate the optimised OSPF weights.

Recall from Chapter 2 that IGPWO is a traditional TE method. It is static, centralised, and works in three phases:

1. Collecting traffic matrix data on a time scale of hours,
2. calculating the optimum IGP weights on a time scale of minutes,
3. and applying the new IGP weights on a time scale of seconds or milliseconds.

In contrast, REPLEX is dynamic, decentralised, and the adaptation process is permanently executed. Thus, a comparison of the convergensce time of REPLEX to the "convergence" time of IGPWO is futile, since the comparison of these times does not make sense. Therefore, we only measure the differences in network performance that can be achieved with IGPWO vs. those that can be achieved by (constantly) running REPLEX.

We take the achieved IP throughput as an indicator for the network performance, as we have seen in the previous experiments that it is closely coupled with the packet loss rate and other performance measures. Let us therefore have another look at some of the remaining lines of Table 7.3 on page 164.

The first line in the first block shows that in the AS 1755 topology, the IGPWO method yields a 10.7 % increase in throughput over the completely unoptimised scenario (i. e., neither IGPWO weights, nor any REPLEX instances running). We can safely assume that this 10.7 % performance increase is not due to some random fluctuation, as the 95 % confidence interval for the mean throughput with normal hop-count routing and the confidence interval for the mean throughput with IGPWO routing are clearly separated. We indicate this fact with the green tick symbol "✔" in the column labelled "*signif.?*".

The second line in the first block shows the results if we use REPLEX as our traffic engineering method instead of IGPWO. We see that in this case, we can achieve an even higher gain of 13.1 % over the completely unoptimised scenario. From the first line of Table 7.4 on the next page, we can see that this corresponds to a 2.2 % gain over the IGPWO simulation. By comparing the confidence intervals for REPLEX and IGPWO however, we see that they actually overlap (marked by "✘" in Table 7.4); we therefore cannot rule out the possibility that the higher value for REPLEX is just a coincidence caused by the randomised workload generator.

The first and second line of the second block in Table 7.3 compare the same values for the topology of AS 3967. Judging by the numbers to those for AS 1755, this topology obviously offers less optimisation potential, but nevertheless we get the same picture

Topology	REPLEX type	TP gain over IGPWO	signif.?
1755	REPLEX	+2.2 %	✘
1755	REPLEX + IGPWO	+0.5 %	✘
1755	REPLEX/noComm	−4.2 %	✔
3967	REPLEX	+1.8 %	✘
3967	REPLEX + IGPWO	+1.0 %	✘
3967	REPLEX/noComm	+0.5 %	✘
3967	REPLEX/noComm + IGPWO	+0.4 %	✘

Table 7.4: IP throughput (TP) gains over traditional IGP weight optimisation (IGPWO) for standard REPLEX, REPLEX without communication (REPLEX/noComm), and REPLEX-IGPWO combinations.

as before: While IGPWO yields a measurable performance improvement and REPLEX seemingly an even better one (Table 7.3), we cannot say with certainty that REPLEX really *is* the better TE mechanism, as the confidence intervals overlap (Table 7.4).

7.4.6 Combining REPLEX and IGP weight optimisation

Remember that we can only apply REPLEX in networks where we can choose between multipath routes. A weight optimisation such as they are performed by IGPWO naturally reduces the number of available routes and therefore the possibilities among which REPLEX instances may choose to share their traffic. Nevertheless, the Totem IGPWO implementation does not necessarily remove all multipath routes from a scenario, but actually features a flag through which one can explicitly allow the use of multipath routes. We enabled this flag, calculated optimised OSPF weights using the IGPWO module, and then let run REPLEX on the network with the optimised OSPF weights.

The result of this combination is shown in the third row of the first and second block of Table 7.3. In both of our simulated topologies, we note a small advantage for the combination of IGPWO and REPLEX over the pure IGPWO setting. Alas, the advantage is rather small and furthermore statistically not significant, as we can see from the second row of the first and second block of Table 7.4. Moreover, the performance gain is less than the one that can be achieved with REPLEX on a network with a simple hop-count routing; on the other hand, this performance difference proved to be not significant either in our simulations.

7.4.7 Disabling communication

Running REPLEX in a network requires additional control traffic for the distance-vector like protocol. Although this traffic is negligibly small in comparison to the real production traffic (especially in Gigabit speed environments and above, as we have shown in Section 6.4.3 on page 110), it is nevertheless interesting to see what happens if we disable this protocol.

In this case, the REPLEX instances do not communicate with each other any more. Rather, they act completely independent, based solely on their local information, which are the measurements on their own outgoing links. This means that a REPLEX instance is unable to detect a downstream bottleneck, unless it is on an immediate outgoing link. Furthermore recall that one single agent from the theoretical model (Section 6.3 ff.) does not correspond to just one single REPLEX instance at one router, but rather to *chains* of REPLEX instances along the agent's paths from the source to the sink (Section 6.4.1 on page 107). If we disable communication between the REPLEX instances, we thus do not faithfully reproduce the actions of our theoretical agents, and our theoretically proven properties (e. g., fast convergence, theorem 6.3.1 on page 106) do not necessarily hold.

agent

instance

Nevertheless, it is interesting to see what happens in a network without any REPLEX communication. First, a network operator may be interested in reducing the additional control traffic to a minimum—and zero definitely *is* a minimum. Second, neighbouring ASes may choose to run REPLEX internally in their respective networks, but without cooperation across the AS boundaries. In this case, we have the situation where two neighbouring routers are equipped with a REPLEX instance each, but where these instances not exchange any information on the traffic conditions in the other network. In both of these cases, if is interesting to see if the crippled "REPLEX" instances still yield performance improvements than a network without REPLEX or with traditional IGPWO traffic engineering.

A first look at the fourth row of the first block in Table 7.3 reveals that we do indeed obtain a measurable performance increase of 5.4 % over the unoptimised case. But this gain is significantly smaller than the ones that can be attained using IGPWO (Table 7.4, first block, third row), or using normal REPLEX with inter-router communication. The difference in performance is large enough to be statistically significant; the confidence intervals do not overlap.

For the 3967 topology, we obtain slightly different results: For this topology, REPLEX without communication leads to a performance improvement (Table 7.3 2nd block, 4th row) that is comparable to the improvements achieved by IGPWO or by normal REPLEX involving communication. The throughput is even half a percent better than the throughput for IGPWO (Table 7.4)—but as before, this difference may be statistically insignificant, as the confidence intervals overlap.

The convergence time also is in the same time scale as in the previous simulations; Fig. 7.27 shows that the convergence as by our definition of convergence takes place within about 250 s (dashed vertical line). The temporal development of the workload throughput takes place on the same timescale or even faster (Fig. 7.28). Due to the smaller gain in comparison to the scenario involving communication, the step following $t = 0$ is less pronounced and thus not as easily discernible. This indiscernibility is even more pronounced for the 3967 topology, as can be seen in Fig. 7.29.

These nice results, however, turn out to be somewhat phony when we take a look at the development of the link utilisation: Fig. 7.30 on page 170 reveals that the maximum link load is actually significantly (and very quickly) increased and not decreased.

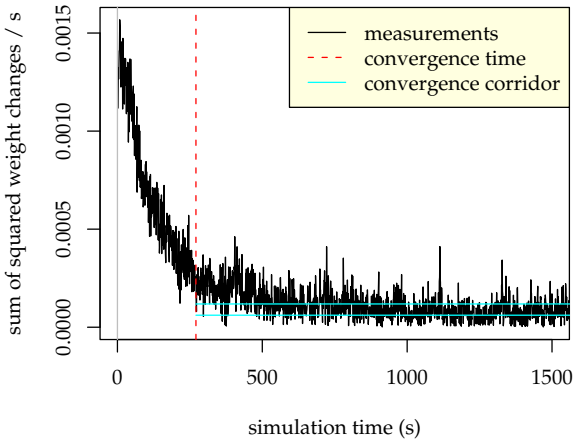


Figure 7.27: Weight changes for REPLEX without communication, AS 1755, standard hop-count routing.

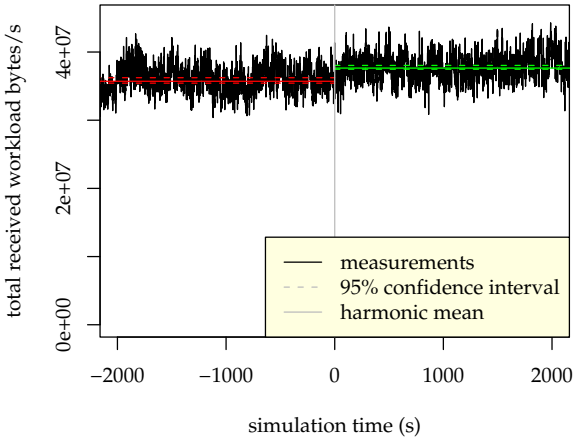


Figure 7.28: Workload throughput for REPLEX without communication, AS 1755, standard hop-count routing.

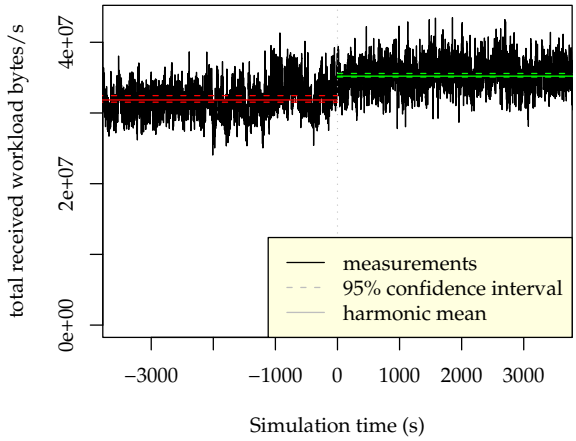


Figure 7.29: Workload throughput for REPLEX without communication, AS 3967, standard hop-count routing.

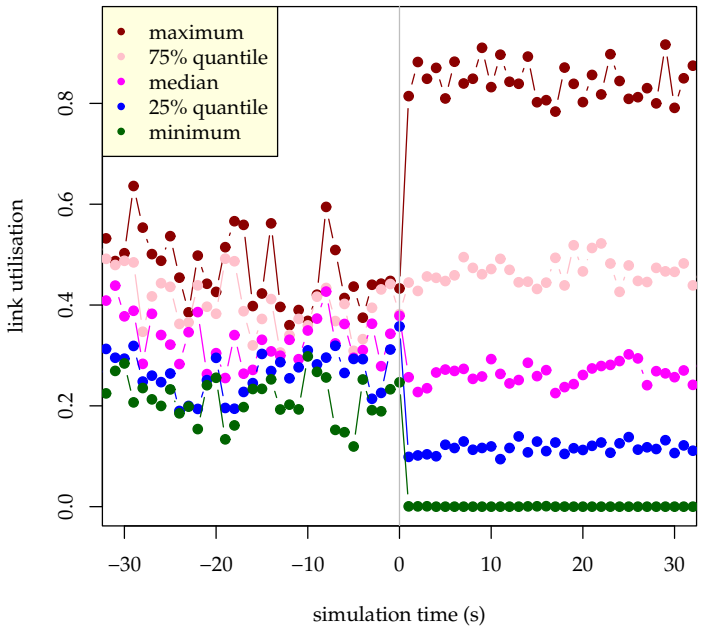


Figure 7.30: Influence of the absence of inter-router communication on the link utilisation distribution in AS 1755.

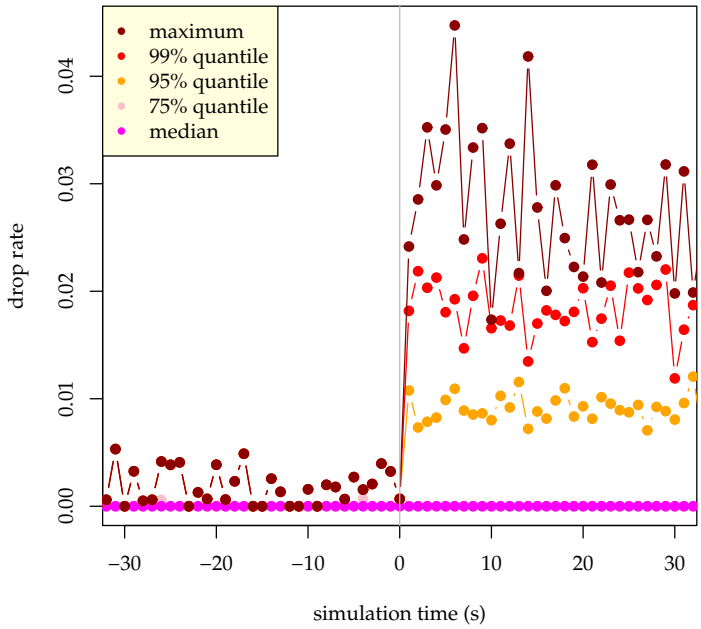


Figure 7.31: Influence of the absence of inter-router communication on the packet loss rate distribution in AS 1755.

Likewise, less loaded links are not used more, but actually less. This unfortunate development is also mirrored in the development of the packet losses (Fig. 7.31): The maximum loss rate is increased, while the quantiles for lower loss rates are decreased. The development for AS 3967 is about the same (plots not shown).

Note that both developments happen very quickly. This is due to the fact that no feedback from other routers is available. Rather, any performance measurement is purely local and thus immediately affects the local weights (in contrast, with REPLEX communication being enabled, the feedback may have to traverse several REPLEX nodes before it reaches a node making a relevant decision; recall the experiments involving the artificial topologies). This also results in a faster convergence time, which is shown in Fig. 7.32 on the next page exemplarily. This plot shows the temporal development of the weight changes for REPLEX without communication in the AS 3267 topology. In contrast to the previously described simulations, we in this simulation apply IGPWO instead of normal hop-count based routing. Regarding the performance, communicationless REPLEX in combination with IGPWO weights delivers a throughput increase of just 0.4 %, which is statistically insignificant in comparison to IGPWO without REPLEX (Table 7.4 on page 167), and yields a performance that is slightly inferior to that of communicationless REPLEX with standard hop-based routing (Table 7.3 on page 164) or any communication-enabled REPLEX scenario (second and third row in second block). However, none of these differences are statistically significant.

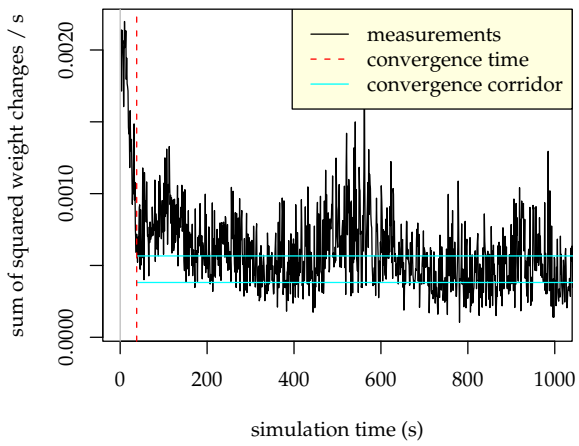


Figure 7.32: Disabling REPLEX communication decreases the convergence time.

The unfortunate situation that already highly congested links become even more congested is caused by the fact that each router only has local knowledge on the its outgoing links. Without receiving additional information from its peers, it cannot know about bottlenecks further downstream. Therefore it is possible that several REPLEX instances decide to shift more traffic towards uncongested links leading to the same router, whose outgoing link towards the destination then of course becomes more congested. Attracting additional traffic this way is especially likely for links that are already highly congested, since these usually are located in the core of the network and thus are well-connected.

We conclude that communicationless REPLEX yields a measurable increase in network throughput. At the same time however, it causes a decrease in the quality of service experienced by some customers, as the maximum link loads and thus the maximum packet losses in the network increase, rather than decrease. This phenomenon also holds for scenarios where we use communicationless REPLEX in combination with IGPWO routing. Therefore, disabling REPLEX communication seems to be a tradeoff whose advantage is questionable, whereas normal REPLEX with inter-router communication proves to be very advantageous.

7.5 Summary

Our simulations involving realistic self-similar traffic with real TCP connections show that REPLEX reliably achieves significant performance improvements. Simple artificial scenarios help us in our search for good REPLEX parameter settings, and show that REPLEX converges quickly to an efficient solution without suffering from oscillations. Simulations with Rocketfuel topologies clearly demonstrate that this also holds in realistic network environments.

When comparing REPLEX against IGPWO, a widespread traditional static engineering technique, our measurements suggest that REPLEX achieves a slightly better performance than IGPWO. However, these differences have so far to be regarded as statistically non-significant.

A strapped-down version without any communication between REPLEX instances on the one hand increases the overall network throughput to some extent as well. On the other hand, it degrades other network performance parameters such as packet loss rates; therefore its effectivity is questionable.

8 Conclusion

In this thesis, we have presented several approaches to characterise and handle Internet traffic and its dynamic behaviour.

We described a methodology that allows us to estimate interdomain demands for Web traffic on a global scale. Our system draws on data from the log files of a content delivery network (CDN), and packet level data for HTTP traffic gathered at one or a few vantage points, and correlates the data. This correlation allows us to extrapolate from the CDN logs to an interdomain traffic matrix for WWW traffic on a truly global scale. Concerning the dynamic behaviour of the traffic, we found that traffic variation on mid-range time scales (1 hour) is less pronounced than on larger time scales (several days).

We then presented a state model for describing the behaviour of users that are browsing the Web using a search engine. Our model captures the temporal and logical relationship between the pages that the users visit during their search sessions, including the search engine itself. A model-based analysis, utilising data from HTTP traces gathered at one vantage point, confirms findings of others, and gives insights into user search behaviour. Finally, it indirectly gives directions for Web site operators, e. g., by showing that redirects leading away from a search hit (i. e., deep link protection) repel many potential visitors.

The previous two approaches are static and require to collect data from potentially many data sources at a single place over a long period of time. We therefore proposed a technique that allows network components, such as routers, to collect data independently, in real time, with very little amounts of memory. Although originally directed at specific computing platforms like network processors, our Expand-and-Collapse algorithm (EaC) is very generic, in that it can be used in any scenario where information is required on the most frequently hit nodes in a search tree, under severe memory constraints. It is thus not even restricted to the context of networking. An evaluation using artificial data shows quick convergence and a good performance.

As the main contribution of the thesis, we finally presented and evaluated REPLEX, a traffic engineering protocol that automatically balances traffic between equal-cost routes provided by an underlying routing architecture, such as OSPF, IS-IS or MPLS. REPLEX scales well even in large networks, since it distributes the required information on current network traffic conditions in a distance vector-like fashion. The protocol-induced communication overhead on a link is only linearly dependent on the number of routes (destination prefixes). Furthermore, our "protocol" even can yield performance increases when communication between the routers is not desired and thus disabled, e. g., in interdomain contexts. Through extensive simulations involving realistic TCP traffic with self-similar properties we first derived reasonable parameter settings

for REPLEX. We then used these parameters in further simulations involving realistic topologies of real ISPs. These simulations show that REPLEX converges quickly, which is in good conformance with its strong game-theoretic background, and that it yields performance improvements that are at least equal to those achieved by traditional offline traffic engineering techniques.

Further analyses and improvements to REPLEX are thinkable. First of all, one can try to improve its performance by applying the weight changes in a "soft" manner (i. e., make a smooth transition to the new weight settings, rather than abruptly setting them to a new value) [KKDC05] so as to further reduce the number of TCP connections affected by shift-induced packet reordering per time interval. A similar effect may be attained by choosing the decision period (T) much smaller than the update interval (T_{comm}).

Another interesting alley is to change the inter-update times on demand: If the performance of a route does not change very much, then the need to tell the neighbours is not as pressing as in a case where a sudden performance increase or decrease is encountered.

We believe that improvements also are achievable by making use of typical flow size distributions [WDF+05]. For example, these can be used together with using improved hashing techniques such as DHFV [JKCM02]. Another approach to utilise the characteristics of flow size distributions can be to aggregate same-egress prefixes such as in [SGD03].

Certainly, providing the REPLEX instances with more path alternatives to select from is another interesting alley. Since hop-count routing already provides a large number of equal-cost multipaths, it is in our view not very promising to find OSPF heuristics for tuning the weights in order to offer greater path flexibility to, e. g., the high-traffic nodes. Instead, we suppose that it is favourable to selectively install MPLS tunnels at specific locations, so as to provide additional route alternatives at specific locations, like, e. g., the SAMTE [SBL06] approach which uses in a combined IGP/MPLS routing.

Furthermore, the performance gain of REPLEX in MPLS-enabled networks in general is another interesting research possibility. In this case, REPLEX needs a slight modification, as now a "link" (i. e., an LSP) spans multiple hosts: Suppose that an LSP consists of the routers $r_0 \rightarrow r_1 \rightarrow \cdots \rightarrow r_n$, where r_0 is the ingress and r_n is the egress. In this case, there is only one REPLEX instance at r_0 that controls the amount of traffic sent via this path to r_n (and in relation to other paths $r_0 \rightarrow r'_1 \rightarrow \cdots \rightarrow r_n$), whereas the other routers $r_1, \ldots$ naturally do not need fully-functional REPLEX instances, because they have no route alternatives. However, they still need to run slimmed-down REPLEX instances, since now the outgoing "link" from r_0 to r_n is not a normal point-to-point link but spans multiple links. As r_0 alone cannot measure the performance (i. e., link load, packet losses, etc.) along the entire LSP, the routers downstream also have to measure their own links that are part of the LSP, and aggregate this data into reports that are sent back to r_0. This aggregation can be made with slightly modified REPLEX instances that only run the distance-vector like protocol without the decision part. Alternatively, it can be interesting to combine REPLEX in a MPLS network together with TeXCP's link

feedback mechanism [KKDC05], since TeXCP also provides performance information for each LSP.

Last but not least, due to its genericity, REPLEX is not restricted to pure IP networks. Instead, it is interesting to see if REPLEX can lead to performance improvements if it is adapted in other networks, e. g., overlays such as peer-to-peer networks, or even completely different types networks that are not even related to computers, e. g., road networks, train networks, or logistics networks. After all, the notion of a Wardrop equilibrium, which forms the theoretical base of REPLEX, was not coined in communication networks, but stems from road traffic research [War52].

Acknowledgements

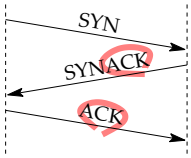


Figure *n*+1: Acknowledgements are a fundamental part of connections.

I am indebted to a number of people—without their support and advice, this work would not have been possible.

The first and by far biggest *thank you* clearly goes to Anja Feldmann, my advisor. As a member of her research group, I got to know how to do research that follows high scientific standards and at the same time covers down-to-earth aspects that are relevant to the industry. She always provided valuable guidance, feedback, and endless support in interesting projects. I hope I have learned some of her mastery in motivating the reader to continue reading the peace of paper he is holding in his hands.

I also learned a lot about scientific writing from my advisor Lukas Kencl during my three-months stay at the Intel Research lab in Cambridge, U. K. (which, to my dismay, has been closed meanwhile—but I deny all responsibility for that...). He also gave valuable feedback and new insights into my presentation methods.

All of my colleagues in our research group at Technische Universität München created a very enjoyable working atmosphere, and definitely increased the pleasure to work here. Furthermore, Robin Sommer, Holger Dreger and Stefan Kornexl helped me a lot whenever I pestered them for taming `bro`. Jörg Wallerich often helped me with strange computer-related problems. Arne Wichmann taught me a number of things about Perl and always had something interesting to say. Manfred Jobmann has always been a source of wisdom whenever I asked him for advice on statistics-related issues, about which I have learned a lot from him. Petra Maier helped a lot in fighting bureaucracy. Olaf Maennel, Vinay Aggarwal, Wolfgang Mühlbauer, Deti Fliegl and, last but not least, Benedikt Elser also provided me with new insights and increased my motivation to come to Garching.

The same holds for my colleagues at Intel Research Cambridge: Without them, my time there would not have been the great time that it was.

It was a pleasure working on REPLEX and its predecessors with Simon Fischer. Not only do I like his sense of humour, but moreover he is a true master in quickly writing good-looking and nicely structured Java code.

Moreover, I want to thank my students, whose Diploma theses, Bachelor theses and system development projects helped me in my work. In particular I want to thank Nataliya Skrypnyuk, Christian Vollmert, and Tobias Schmidbauer for their implementations, and Stefan Schneider for his ambitious Web traffic analysis.

Last but not least, I am very thankful to Thomas Fuhrmann, Bruce Maggs, and Christian Scheideler, for being part of my committee—which implied tasks such as reading this 200 page thesis writing reports, organising and attending the *rigorosum*, etc.

Даниела, благодаря ти много за твоята любов.

And above all, I thank my parents for their love and support in all these years.

List of Figures

Bibliography

[AC03] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proceedings of ACM SIGCOMM Conference*, 2003.

[ACK03] S. Agarwal, C.-N. Chuah, and R. Katz. OPCA: Robust interdomain policy routing and traffic control. In *IEEE Openarch*, April 2003.

[AdMD+04] Frank-Uwe Andersen, Hermann de Meer, Ivan Dedinski, Tobias Hossfeld, Cornelia Kappler, Andreas Mäder, Jens O. Oberender, and Kurt Tutschku. An architecture concept for mobile P2P file sharing services, 2004.

[AFM05] Vinay Aggarwal, Anja Feldmann, and Sebastian Mohr. Implementation of a P2P system within a network simulation framework. In *Proceedings of the European Conference on Complex Systems (ECCS)*, 2005.

[AK04] Baruch Awerbuch and Robert D. Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proceedings of 36th Annual ACM Symposium on Theory of Computing (STOC)*, 2004.

[Aka] Akamai. `http://www.akamai.com`.

[AMA+99] D. O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for traffic engineering over MPLS. RFC 2702, 1999.

[AMSW] Tom Anderson, Ratul Mahajan, Neil Spring, and David Wetherall. Rocketfuel: An ISP topology mapping engine. `http://www.cs.washington.edu/research/networking/rocketfuel/`.

[AS03] L. Andersson and G. Swallow. The Multiprotocol Label Switching (MPLS) Working Group decision on MPLS signaling protocols, 2003. RFC 3468 (and follow-ups).

[AW97] M. F. Arlitt and C. L. Williamson. Internet Web servers: Workload characterization and implications. *IEEE/ACM Transactions on Networking*, 1997.

[Awd99] D. O. Awduche. MPLS and traffic engineering in IP networks. *IEEE Communication Magazine*, 1999.

[Awd02] Awduche, D. et al. Overview and principles of Internet traffic engineering. Request for Comments 3272, 2002.

[AWS06] Richard Atterer, Monika Wnuk, and Albrecht Schmidt. Knowing the user's every move—user activity tracking for website usability evaluation and implicit interaction. In *WWW*, 2006.

[BÖ3] Hagen Böhm. Analysis of OSPFv2–BGP4 interactions using the SSFNet simulator. Master's thesis, Universität des Saarlandes, October 2003.

[Bar01] Paul Barford. *Modeling, Measurement and Performance of World Wide Web Transactions*. PhD thesis, Boston University, 2001.

[BC98] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS Conference*, pages 151–160, 1998.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM*, 1999.

[BEDL05] Avrim Blum, Eyal Even-Dar, and Katrina Ligett. Routing without regret, 2005. Manuscript.

[BFG+06] Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, Zengjian Hu, and Russel Martin. Distributed selfish load balancing. In *Proceedings of 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

[BJC+04] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *ACM SIGIR*, 2004.

[BK05] Andrea Bergamini and Lukas Kencl. Network of Shortcuts: An adaptive data structure for tree-based search methods. In *Proceedings of IFIP Networking*, Waterloo, Canada, May 2005.

[BLM] S. Balon, J. Lepropre, and G. Monfort. TOTEM—TOolbox for Traffic Engineering Methods. `http://totem.run.montefiore.ulg.ac.be/`.

[BLR03] A. Basu, A. Lin, and S. Ramanathan. Routing using potentials: A dynamic traffic-aware routing algorithm. In *Proceedings of ACM SIGCOMM Conference*, 2003.

[BMW56] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics and Transportation*. Yale University Press, 1956.

[Bog] Alex Bogomolny. Cut the knot! — the constitution and paradoxes. `http://www.cut-the-knot.org/ctk/Democracy.shtml`. viewed on Jan 26th, 2007.

[BPS+98] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. TCP behavior of a busy Internet server: Analysis and improvements. In *Proceedings of IEEE INFOCOM*, 1998.

[BV02] L. Bent and G. M. Voelker. Whole page performance. In *In Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, 2002.

[BZB+97] P. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP)—version 1 functional specification, 1997. RFC 2205.

[CB96] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and causes. In *Proceedings of ACM SIGMETRICS*, 1996.

[CB97] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 1997.

[CFS05] Michael Chau, Xiao Fang, and Olivia R. Lui Sheng. Analysis of the query logs of a web site search engine. In *American Society for Information Science and Technology*, 2005.

[Cis] Cisco Netflow. `http://www.cisco.com/warp/public/732/netflow/index.html`.

[CJMW05] Hyunseok Chang, Sugih Jamin, Zhuoqing Morley Mao, and Walter Willinger. An empirical approach to modeling inter-AS traffic matrices. In *Proceedings of ACM Measurement Conference*, Berkeley, California, USA, 2005. ACM and USENIX.

[Cla04] B. Claise. Cisco Systems NetFlow services export version 9, 2004. RFC 3954.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[CMPS02] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and applicability statements for internet standard management framework, 2002. RFC 3410 (and follow-ups).

[CWNM03] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Query expansion by mining user logs. In *IEEE Trans. Knowl. Data Eng. 15(4)*, 2003.

[CWZ00] Z. Cao, Z. Wang, and E. W. Zegura. Performance of hashing-based schemes for Internet load balancing. In *Proceedings of IEEE INFOCOM Conference*, 2000.

[Dav03] B. D. Davison. Content delivery and distribution services, 2003. `http://www.web-caching.com/cdns.html`.

[DBCP97] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. *ACM Computer Communication Review*, 27(4):3–14, October 1997.

[DFM+06] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Proceedings of 15th Usenix Security Symposium*, 2006.

[Dir] Google Directory. TCP implementations. `http://www.google.com/Top/Computers/Internet/Protocols/Transmission_Protocols/TCP/Implementations/`, accessed on Nov. 4th, 2007.

[DMP+02] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 2002.

[EDM05] Eyal Even-Dar and Yishay Mansour. Fast convergence of selfish rerouting. In *Proceedings of 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[EJLW01] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proceedings of IEEE INFOCOM Conference*, 2001.

[EKMV04] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *Proceedings of ACM SIGCOMM*, Portland, OR, USA, September 2004.

[FBR03] N. Feamster, J. Borkenhagen, and J. Rexford. Guidelines for interdomain traffic engineering. In *Proceedings of ACM SIGCOMM*, 2003.

[Fel00a] A. Feldmann. BLT: Bi-layer tracing of HTTP and TCP/IP. In *Proceedings of WWW-9*, 2000.

[Fel00b] A. Feldmann. Characteristics of TCP connection arrivals. In K. Park and W. Willinger, editors, *Self-Similar Network Traffic And Performance Evaluation*. J. Wiley & Sons, Inc. 2000.

[FGHW99] Anja Feldmann, Anna Gilbert, Polly Huang, and Walter Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *ACM SIGCOMM Conference*, September 1999.

[FGL+00a] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. NetScope: Traffic engineering for IP networks. *IEEE Network Magazine*, 2000.

[FGL+00b] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *Proceedings of ACM SIGCOMM*, 2000.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol — HTTP/1.1, 1999. RFC 2616.

[FKF06] Simon Fischer, Nils Kammenhuber, and Anja Feldmann. REPLEX—dynamic traffic engineering based on Wardrop routing policies. In *Proceedings of CoNext*, Lisboa, Portugal, 2006.

[FM00] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proceedings of IEEE Infocom*, 2000.

[FP99] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proceedings of IEEE Global Internet*, 1999.

[FRT02] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. In *IEEE Communication Magazine*, pages 118–124, 2002.

[FRV06] Simon Fischer, Harald Räcke, and Berthold Vöcking. Fast convergence to Wardrop equilibria by adaptive sampling methods. In *Proceedings of 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 653–662, Seattle, WA, USA, May 2006.

[FT00] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings of IEEE INFOCOM Conference*, 2000.

[FT02] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. In *IEEE Journal on Selected Areas in Communications, Vol. 20, No. 4*, 2002.

[FV04] Simon Fischer and Berthold Vöcking. On the evolution of selfish routing. In Susanne Albers and Tomasz Radzik, editors, *Proceedings of 12th Annual European Symposium on Algorithms (ESA)*, number 3221 in Lecture Notes in Computer Science, Bergen, Norway, September 2004. Springer-Verlag.

[FV05] Simon Fischer and Berthold Vöcking. Adaptive routing with stale information. In Marcos Kawazoe Aguilera and James Aspnes, editors, *Proceedings of 24th Annual ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing (PODC)*, Las Vegas, NV, USA, 2005.

[FW97] A. Feldmann and W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. In *Proceedings of IEEE Infocom*, 1997.

[Gao00] L. Gao. On inferring autonomous system relationships in the Internet. In *Proceedings of IEEE Global Internet*, 2000.

[GB97] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[GCR01] S. Gadde, J. S. Chase, and M. Rabinovich. Web caching and content distribution: a view from the interior. *Computer Communications*, 2001.

[Ggl] Google basic search. `http://www.google.com/support/bin/static.py?page=searchguides.html&ctx=basics`.

[GH98] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley, 1998.

[GM99] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Proceedings of Hot Interconnects VII*, 1999.

[GP01] T. G. Griffin and B. J. Premore. An experimental analysis of BGP convergence time. In *Proceedings of International Conference on Network Protocols*, 2001.

[GR97] R. Govindan and A. Reddy. An analysis of Internet inter-domain topology and route stability. In *Proceedings of IEEE INFOCOM*, 1997.

[GW99] T. G. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *Proceedings of ACM SIGCOMM*, 1999.

[GZRR03] Ivan Gojmerac, Thomas Ziegler, Fabio Ricciato, and Peter Reichl. Adaptive multipath routing for dynamic traffic engineering. In *Proceedings of GLOBECOM*, 2003.

[Hal97] B. Halabi. *Internet Routing Architectures*. Cisco Press, 1997.

[HBCR07] Jiayue He, Ma'ayan Bresler, Mung Chiang, and Jennifer Rexford. Towards multi-layer traffic engineering: Optimization of congestion control and routing. *IEEE Journal on Selected Areas in Communications*, 2007.

[HDA05] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer TCP throughput. In *Proceedings of ACM SIGCOMM*, 2005.

[Hen] Tom Henderson. Optical density and light speed. `http://www.glenbrook.k12.il.us/gbssci/Phys/Class/refrn/u14l1d.html`.

[HI05] Wang Hao and Robert Ito. Dynamics of load-sensitive adaptive routing. In *Proceedings of IEEE International Conference on Communications (ICC)*, 2005.

[Hul02] S. Hull. *Content Delivery Networks: Web Switching for Security, Availability, and Speed*. McGraw-Hill, 2002.

[IDGM01] G. Iannaccone, C. Diot, I. Graham, and N. McKeown. Monitoring very high speed links. In *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, USA, November 2001.

[ISZ99] A. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 1999.

[IUKB+04] M. Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, P. A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. *Lecture Notes in Computer Science*, 3015, 2004.

[JAC+02] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, and A. Malis. Constraint-based LSP setup using LDP, 2002. RFC 3212.

[JCDK00] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, 2000.

[JK03] E. J. Johnson and A. R. Kunze. *IXP2400/2800 Programming*. Intel Press, 2003.

[JKCM02] J.-Y. Jo, Y. Kim, H. J. Chao, and F. Merat. Internet traffic load balancing using dynamic hashing with flow volume. In *Proceedings ofSPIE ITCom*, 2002.

[JP01] B. Jansen and U. Pooch. Web user studies: A review and framework for future work. In *American Society of Information Science and Technology*, 2001.

[KB02] L. Kencl and J.-Y. Le Boudec. Adaptive load sharing for network processors. In *Proceedings of IEEE Infocom*, New York, 2002.

[KBB+04] Karagiannis, Broido, Brownlee, k. c. claffy, and Faloutsos. Is P2P dying or just hiding? In *Proceedings of Globecom*, 2004.

[KKDC05] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: responsive yet stable traffic engineering. In *Proceedings of ACM SIGCOMM*, 2005.

[KKV+03] M. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. Campbell. Directions in packet classification for network processors. In *Proceedings of the 2nd IEEE Workshop on Network Processors*, 2003.

[KKY03] D. Katz, K. Kompella, and D. Yeung. Traffic engineering extensions to OSPF version 2. Internet Draft, 2003.

[KLM97] T. M. Kroeger, D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[KR01] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice*. Addison-Wesley, 2001.

[Kra03] Thomas Kraemer. IP traffic engineering—OSPF vs. MPLS. Master's thesis, Universität des Saarlandes, 2003.

[KS06] Lukas Kencl and Christian Schwarzer. Traffic-adaptive packet filtering of denial of service attacks, 2006.

[KV05] Frank Kelly and Thomas Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM Computer Communication Review*, 35(2):5–12, 2005.

[KWZ01] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings of ACM Measurement Workshop*, 2001.

[KZ89] A. Khanna and J. Zinky. The revised ARPANET routing metric. In *Proceedings of ACM SIGCOMM Conference*, 1989.

[LAJ99] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and wide-area network failures. In *Proceedings of International Symposium on Fault-Tolerant Computing*, 1999.

[LBBSS02] Nathaniel Leibowitz, Aviv Bergman, Roy Ben-Shaul, and Aviv Shavit. Are file swapping networks cacheable?—Characterizing P2P traffic. In *Proceedings of 7th IWCW*, 2002.

[LG02] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, September/October 2002.

[Liu] Bob Liu. A different approach to content delivery. `http://www.isp-planet.com/news/2001/routescience.html`.

[LLC05] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *WWW*, 2005.

[LMJ98] C. Labovitz, R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 1998.

[LMJ99] C. Labovitz, R. Malan, and F. Jahanian. Origins of Internet routing instability. In *Proceedings of IEEE INFOCOM*, 1999.

[LN] B. Lavoie and H.F. Nielsen. Web characterization terminology & definitions sheet. `http://www.w3c.org/1999/05/WCA-terms/`.

[LR07] Yong Liu and A. L. Narasimha Reddy. Multihoming route control among a group of multihomed stub networks. *Computer Communications*, 30(17):3335–3345, November 2007.

[LTWW94] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic. *IEEE/ACM Transactions on Networking*, 1994.

[LW04] Julia Luxenburger and Gerhard Weikum. Query-log based authority analysis for web information search. In *WISE*, 2004.

[LY03] G. Liang and B. Yu. Pseudo likelihood estimation in nework tomography. In *Proceedings of IEEE INFOCOM*, March 2003.

[MA99] W. Stevens M. Allman, V. Paxson. TCP congestion control, 1999. RFC 2581.

[MFT+02] A. Medina, C. Fraleigh, N. Taft, S. Bhattacharyya, and C. Diot. A taxonomy of IP traffic matrices. In *Workshop on Scalability and Traffic Control in IP Networks at the SPIE ITCOM+OPTICOMM Conference*, 2002.

[MGWR01] D. McPerson, V. Gill, D. Walton, and A. Retana. BGP persistent route oscillation condition, 2001. Internet Draft (draft-ietf-idr-route-oscillation-01.txt).

[MHH+03] A. Moore, J. Hall, E. Harris, C. Kreibich, and I. Pratt. Architecture of a network monitor. In *Proceedings of the Fourth Passive and Active Measurement (PAM) Workshop*, April 2003.

[Mir] `http://www.mirror-image.com`.

[MM02] Petar Maymunkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[MTS+02] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic Matrix Estimation: Existing Techniques and New Directions. In *Proceedings of ACM SIGCOMM*, 2002.

[Nag84] John Nagle. Congestion control in IP/TCP internetworks, 1984. RFC 896.

[NDE+04] N. Nyakoya, D. Dineva, F. Ebener, B. Elser, G. Groh, D. Krick, F. Müller, L. Raber, P. Rullmann, C. Stahl, and P. Walter. On the unlikeliness of detecting bogus entries in bibliographies. In *Proceedings of the 2nd International Conference on General Aspects in Computer Science*, Ostrov Kozloduy, Bulgaria, February 2004.

[Net] CPAN-Comprehensive Perl Archive Network. `HTML::Parser`. `http://www.cpan.org/`.

[NK98] S. Nilsson and G. Karlsson. Fast address lookup for Internet routers. In *Proceedings IFIP 4th International Conference on Broadband Communications '98*, pages 11–22, 1998.

[PAD+99] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known TCP implementation problems, 1999. RFC 2525.

[PAMM98] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP performance metrics. Request for Comments 2330, 1998.

[Pax99] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, volume 31, no. 23–24, pages 2435–2463, 1999.

[Pea81] J. Postel et al. RFC 793, September 1981. `http://www.ietf.org/rfc/rfc793.txt`.

[PF95] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 1995.

[PM95] V. N. Padmanabhan and J. C. Mogul. Improving HTTP latency. *Computer Networks and ISDN Systems*, 1995.

[Pop] Adrian Popa. What is the speed of light in a fiber-optic cable? `http://www.madsci.org/posts/archives/feb2001/983369337.Ph.r.html`.

[Pos81] Jon Postel. Internet Protocol, 1981. RFC 791.

[PTD04] K. Papagiannaki, N. Taft, and C. Diot. Impact of flow dynamics on traffic engineering design principles. In *Proceedings of IEEE Infocom*, Hong Kong, 2004.

[PUB04] Cristel Pelsser, Steve Uhlig, and Olivier Bonaventure. On the difficulty of establishing interdomain LSPs. In *IEEE IPOM*, 2004.

[PW00] Kihong Park and Walter Willinger, editors. *Self-Similar Network Traffic and Performance Evaluation*. Wiley-Interscience, 2000.

[Qea03] B. Quoitin and et al. Interdomain traffic engineering with BGP. *IEEE Communications Magazine, Internet Technology Series*, 2003.

[QUB02] B. Quoitin, S. Uhlig, and O. Bonaventure. Using redistribution communities for interdomain traffic engineering. In *Quality of Future Internet Services (QoFIS 2002)*, 2002.

[QYZS03] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker. On selfish routing in internet-like environments. In *Proceedings of ACM SIGCOMM*, pages 151–162, Karlsruhe, Germany, August 2003.

[Ren] Renesys. The SSFNet network simulator. `http://www.ssfnet.org/`. Java edition, version 2.0.

[RGK+01] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. In *Proceedings of ACM Measurement Workshop*, 2001.

[RIP] RIPE's Routing Information Service Raw Data Page. `http://data.ris.ripe.net/`.

[RJ05] Filip Radlinski and Thorsten Joachims. Query chains: Learning to rank from implicit feedback. In *KDD*, 2005.

[RK04] V. Raghunathan and P. R. Kumar. A Wardrop routing protocol for ad hoc wireless networks. In *IEEE CDC*, 2004.

[RK05] V. Raghunathan and P. R. Kumar. Issues in Wardrop routing in wireless networks. In *IEEE WICON*, 2005.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4), 2006. RFC 4271.

[Rou] RouteViews project. `http://www.routeviews.org/`.

[RT02] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2), 2002.

[RTZ03] M. Roughan, M. Thorup, and Y. Zhang. Traffic engineering with estimated traffic matrices. In *Proceedings of ACM Measurement Conference*, 2003.

[RV03] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, USA, June 2003.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture, 2001. RFC 3031.

[RWXZ02] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *Proceedings of ACM Measurement Workshop*, 2002.

[SBL06] F. Skivée, S. Balon, and G. Leduc. A scalable heuristic for hybrid IGP/MPLS traffic engineering—Case study on an operational network. In *Proceedings of 14th IEEE International Conference on Networks*, Singapore, September 2006. IEEE Xplore.

[SBVW03] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[Sch06] Stefan Schneider. Automated deriving of statistical parameters characterizing network traffic. Systementwicklungsprojekt (systems development project) report, 2006.

[SD98] R. Hinden S. Deering. Internet Protocol, version 6 (IPv6) specification, 1998. RFC 2460.

[SDV02] A. Shaikh, R. Dube, and A. Varma. Avoiding instability during graceful shutdown of OSPF. In *Proceedings of IEEE INFOCOM*, 2002.

[SF02] R. Sommer and A. Feldmann. Netflow: Information loss or win? In *Proceedings of ACM Measurement Workshop*, 2002.

[SGD+02] S. Saroiu, K. Gummadi, R. J. Dunn, S. Gribble, and H. Levy. An analysis of Internet content delivery systems. In *Proceedings of OSDI*, 2002.

[SGD03] Ashwin Sridharan, Roch Guérin, and Christophe Diot. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks. In *Proceedings of IEEE INFOCOM Conference*, 2003.

[SHMM98] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz. Analysis of a very large AltaVista query log. Technical Report Technical Note 014, SRC, 1998.

[SJO00] Amanda Spink, Bernard J. Jansen, and H. Cenk Ozmultu. Use of query reformulation and relevance feedback by excite users. In *Internet Research: Electronic Networking Applications and Policy*, 2000.

[SKDV00] Aman Shaikh, Lampros Kalampoukas, Rohit Dube, and Anujan Varma. Routing stability in congested networks: Experimentation and analysis. In *Proceedings of ACM SIGCOMM*, 2000.

[SKP+05] Amanda Spink, Sherry Koshman, Minsoo Park, Chris Field, and Bernard J. Jansen. Multitasking web search on `vivisimo.com`. In *ITCC*, 2005.

[Skr06] Nataliya Skrypnyuk. Load-sensitive routing. Master's thesis, Technische Universität München, 2006.

[SMW02] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of ACM SIGCOMM Conference*, Pittsburgh, PA, USA, August 2002. ACM.

[SNL+04] A. Soule, A. Nucci, E. Leonardi, R. Cruz, and N. Taft. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *Proceedings of ACM SIGMETRICS*, 2004.

[SR06] Daniel Stutzbach and Reza Rejaie. Understanding churn in P2P networks. In *Proceedings of Proceedings of ACM Measurement Conference*, 2006.

[SRS99] Anees Shaikh, Jennifer Rexford, and Kang Shin. Load-sensitive routing of long-lived IP flows. In *Proceedings of ACM SIGCOMM*, 1999.

[STZ05] Xuehua Shen, Bin Tan, and Cheng Xiang Zhai. Context-sensitive information retrieval using implicit feedback. In *ACM SIGIR*, 2005.

[SWJS01] Amanda Spink, Dietmar Wolfram, B. J. Jansen, and Tefko Saračević. Searching the web: The public and their queries. In *American Society for Information Science and Technology*, 2001.

[SZR06] Daniel Stutzbach, Shanyu Zhao, and Reza Rejaie. Characterizing files in the modern gnutella network. In *Multimedia Computing and Networking*, 2006. extended paper version, `http://mirage.cs.uoregon.edu/pub/stutzbach-2006-msj.pdf`.

[TDRR05] Renata Teixeira, Nick Duffield, Jennifer Rexford, and Matthew Roughan. Traffic matrix reloaded: Impact of routing changes. In *Proceedings of Passive and Active Measurement*, 2005.

[Tei05] Renata Teixeira. *Network Sensitivity to Intradomain Routing Changes*. PhD thesis, University of California San Diego, August 2005.

[TMW97] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network Magazine*, 1997.

[Tut04] Kurt Tutschku. A measurement-based traffic profile of the eDonkey filesharing service. *Lecture Notes in Computer Science*, 3015, 2004.

[UBQ03] S. Uhlig, O. Bonaventure, and B. Quoitin. Interdomain traffic engineering with minimal BGP configurations. In *18th International Teletraffic Congress (ITC)*, September 2003.

[UQLB06] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM CCR*, 36(1):83–86, 2006.

[Vol04] Christian Vollmert. A Web workload generator for the SSFNet network simulator. Bachelor's thesis, Technische Universität München, 2004.

[VR02] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer Verlag, 4th edition, 2002.

[Wal01] Jörg Wallerich. Design and implementation of a WWW workload generator for the ns-2 network simulator. Master's thesis, Universität des Saarlandes, 2001.

[Wal03] Philipp Walter. Evaluation of random early detection router queues. Master's thesis, Universität des Saarlandes, 2003.

[War52] John Glen Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institute of Civil Engineers, Pt. II*, 1952.

[WDF+05] Jörg Wallerich, Holger Dreger, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. A methodology for studying persistency aspects of internet flows. *ACM SIGCOMM CCR*, 2005.

[Wik07a] Wikipedia. Apportionment (politics). `http://en.wikipedia.org/w/index.php?title=Apportionment_(politics)&oldid=102173298`, January 2007.

[Wik07b] Wikipedia. Sitzzuteilungsverfahren. `http://de.wikipedia.org/w/index.php?title=Sitzzuteilungsverfahren&oldid=25652212`, January 2007.

[Wik07c] Wikipedia. Termination proof. `http://en.wikipedia.org/w/index.php?title=Termination_proof&oldid=168111946`, October 2007.

[WJR02] J. Winick, S. Jamin, and J. Rexford. Traffic engineering between neighboring domains. `http://www.research.att.com/~jrex/papers/interAS.pdf`, July 2002.

[WM00] C. E. Wills and M. Mikhailov. Studying the impact of more complete server information on Web caching. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, 2000.

[WOHM06] Harald Weinreich, Hartmut Obendorf, Eelco Herder, and Matthias Mayer. Off the beaten tracks: Exploring three aspects of web navigation. In *WWW*, 2006.

[WPT98] W. Willinger, V. Paxson, and M. S. Taqqu. Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, 1998.

[WVTP97] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP route lookups. In *Proceedings of ACM SIGCOMM*, Cannes, France, 1997.

[WXQ+06] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. COPE: Traffic engineering in dynamic networks. In *Proceedings of ACM SIGCOMM Conference*, 2006.

[XHBN00] X. Xiao, A. Hannan, B. Bailey, and L. Ni. Traffic engineering with MPLS in the Internet. In *IEEE Network Magazine*, March 2000.

[XR06] Wen Xu and Jennifer Rexford. Miro: Multi-path interdomain routing. In *Proceedings of ACM SIGCOMM Conference*, 2006.

[YQL04] Xian Yu, Chunming Qiao, and Yona Liu. TCP implementations and false time out detection in OBS networks. In *Proceedings of IEEE INFOCOM*, 2004.

[ZGK+05] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Towsley. Optimal routing with multiple traffic matrices: Tradeoff between average case and worst case performance. In *Proceedings of 13th International Conference on Network Protocols (ICNP)*, 2005.

[Zip] Zipf's law. `http://linkage.rockefeller.edu/wli/zipf`.

[ZRDG03] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proceedings of ACM SIGMETRICS*, 2003.

[ZRLD03] Yin Zhang, Matthew Roughan, Carsten Lund, and David Donoho. An information-theoretic approach to traffic matrix estimation. In *Proceedings of ACM SIGCOMM Conference*, 2003.

Glossary

Akamai: A big[1] CDN provider. Provides Web (and other) services.

CDN: Content Delivery Network. A collection of servers are distributed across the entire Internet. Typically, all servers provide the same content. The idea is that a client uses a server that is nearby, thereby experiencing a performance improvement.

distance vector protocol: A distance vector protocol (short DV) is a routing protocol that performs a distributed computation of the Bellman-Ford algorithm. Every node sends messages to its neighbours, containing the distances to all nodes for which it has a route. The receivers include this information into their routing tables and eventually change them accordingly. Every time that the routing table changes (be it through reception of such a message, or through a new link being added, or an existing link going down), a router immediately informs its neighbours.

ISP: Internet Service Provider

link state protocol: A routing protocol based on full information. Every node receives an exhaustive map of the network topology, i. e., a graph showing which nodes are connected to which other nodes. Each node then independently calculates the best next hop from this map for every possible destination in the network, using only the copy of the map, and without any further communication with any other node. The topology map usually is obtained through broadcasts.

MSS: Maximum Segment Size, i. e., the maximum number of bytes that is allowed in a (TCP) packet.

RTT: Round trip time, i. e., the sum of the one-way delay from the sender to the receiver and the one-way delay back from the receiver to the sender. As asymmetric routing is rather common in the Internet, this is in many *not* just the double of one of the one-way delays.

traffic engineering: Optimising the performance of a network, usually achieved by making adjustments to the routing based on measured (or estimated) traffic demands.

[1] (at the time of this writing)

Zipf: A **Zipf-like distribution** is a probability distribution where the contribution of the k-th most popular item varies as $\frac{1}{k^a}$, for some a. Values taken from a Zipf distribution normally form a linear slope in a doubly-logarithmic CCDF plot.

Index