Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kommunikationsnetze

Analysis and Design of Distributed Control Plane Mechanisms in SDN-based Industrial Networks

Ermin Sakic, M.Sc.

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Sebastian Steinhorst
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Wolfgang Kellerer
	2.	Prof. Dr.-Ing. Hermann de Meer

Die Dissertation wurde am 17.06.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 14.12.2020 angenommen.

Analysis and Design of Distributed Control Plane Mechanisms in SDN-based Industrial Networks

Ermin Sakic, M.Sc.

14.12.2020

Abstract

This thesis analyzes the existing work, proposes, and evaluates novel designs surrounding the logically-centralized, physically-distributed Software Defined Networking (SDN) network control plane in industrial settings.

The motivation behind the presented work is two-fold. First, the recent rise in adoption of SDN in data-center and campus networks has sparked an interest in the corresponding technology transfer to the industrial domain. While the application of SDN for industrial use cases is well understood, necessary adaptation of SDN designs to fulfill the non-functional requirements and constraints of the industrial domain remains an open point. Second, recently proposed Ethernet extensions by the Institute of Electrical and Electronics Engineers (IEEE) 802.1 Time-Sensitive Networking (TSN) group improve the determinism of performance of distributed real-time applications, at the expense of an added network configuration complexity. Efficient management of the proposed TSN mechanisms can only be achieved by a centralized decision-making entity, equipped with admission control, scheduling, and planning logic. Future industrial networks will require an efficient interplay of data and control plane in order to enable strict End-To-End (E2E) latency, control plane response time, multi-tenancy and auto-configuration requirements imposed by the dynamic Industry 4.0 use cases. Combined with the prerequisite of robust operation, novel control plane designs are necessary in order to enable its guaranteed low response time while minimizing the associated deployment complexity.

Indeed, deployment of SDN in industrial scenarios requires catering for the issue of control plane dependability. The impact of distributed controller operation on resulting network performance is, however, under-investigated in existing literature. A provably robust logically-centralized industrial control plane design is necessary for its successful adoption in production settings.

We postulate the feasibility of using a highly-available and resilient SDN controller solution as an enabler of future softwarized industrial networks. To this end, we provide an analysis of the availability, reliability and response time properties of the existing consensus-based solutions. In order to achieve the low response times, we propose multiple enhancements to handling flexibly-consistent control state updates at scale. We furthermore define mechanisms for tolerating semantic faults in replicated controller state independent of the root cause (e.g., software / hardware bug, malicious takeover or diverged controller state). The proposed designs are validated analytically and empirically. To simplify the deployment of the resulting control plane, we propose a novel automated bootstrapping approach that omits any data plane dependencies, so to isolate the control and data plane responsibilities, providing for easier verification and analysis of the system's correctness.

Succinctly summarized, our thesis achieves four goals:

- *Assessment and hardening of existing distributed SDN control plane designs*: We provide the analytical guarantees for availability and response time metrics of state-of-the-art distributed SDN control plane proposals. Steady-state and transient analysis based on Stochastic Activity Networks (SANs) are used in dependability and performance evaluation. We furthermore assess corner cases impacting the correctness of existing control plane designs. In particular, scenarios of leader oscillation and unsuccessful election were reproduced with existing SDN controllers. To cater for and alleviate such issues, we propose for decoupling of the underlying failure detection procedure from controller state consensus.

- *Design of a scalable fault-tolerant distributed control plane*: We propose multiple designs for realizing a multi-controller SDN control plane that simultaneously enables a *Fail-Stop*-tolerant and scalable system operation. To this end, we introduce the notion of *adaptive consistency*, a state replication model that autonomously adapts to provide for a *sufficient* degree of consistency for the hosted SDN applications, under consideration of the constraints on the worst-case state divergence.

- *Design of mechanisms for supporting reliable distributed control plane operation*: To ensure correct handling of faults rooted in Byzantine events, we propose novel control mechanisms that guarantee a transparent system transition from faulty-to-stable state even if some controller replicas are computing unreliable outputs due to internal faults. The proposed control plane extensions optionally leverage programmable forwarding elements in order to minimize the footprint of controller instance replication.

- *Automated bootstrapping of a highly-available and reliable distributed control plane*: We propose two novel bootstrapping schemes to initialize a complex distributed system comprising arbitrary number of controller replicas. The in-band control plane is thus bootstrapped with availability guarantees - i.e., it is automatically protected against individual data plane and controller failures.

The majority of designs proposed in this thesis were evaluated under assumption of industrial network Key Performance Indicators (KPIs), i.e., they assume the respective typical topologies and parameter configurations. Nevertheless, the advantages of introduced designs apply to other domains, e.g., the data-center and campus SDNs.

Kurzfassung

Diese Arbeit liefert eine gründliche Analyse des Standes der Technik im Kontext der verteilten SDN-Netzsteuerungsebene. Desweiteren schlägt sie neuartige Funktionen und Entwürfe für industrielle SDN-Netzsteuerungsebene vor und führt empirische und analytische Auswertungen der resultierenden Entwürfen in realen Umgebungen durch.

Die Motivation hinter der vorgestellten Arbeit ist zweifach. Erstens hat die Adoption von SDN in Rechenzentrums- und Campusnetzen ein Interesse am entsprechenden Technologietransfer in den industriellen Bereich geweckt. Während die generellen Anwendungsfälle von SDN im industriellen Kontext umfassend untersucht sind, werden weitere Anpassungen und Erweiterungen der kritischen Steuerungsfunktionen benötigt, die ebenso notwendigen, nicht-funktionalen Anforderungen umzusetzen. Zweitens, die jüngst verabschiedenten Erweiterungen der Ethernetstandards in der IEEE TSN gruppe erzielen die Sicherstellung der deterministischen Kommunikation der verteilten Echtzeitanwendungen, allerdings passiert dies zugunsten der erhöhten Komplexität in Netzkonfiguration. Eine effiziente Verwaltung der vorgeschlagenen TSN-Mechanismen kann nämlich nur von einer zentralen Entscheidungseinheit erreicht werden, die mit Mechanismen für Zugangskontrolle und Ablaufplanungslogik ausgestattet ist, die aber den Anforderungen der hohen Zuverlässigkeit bedarf.

Die Realisierung zukünftiger Industrienetze erfordert damit ein effizientes Zusammenspiel zwischen Daten- und Steuerungsebenen des SDNs um die strikten Anforderungen aus der Industrie 4.0 zu ermöglichen u.a. auch begrenzte E2E Paketverzögerungen, niedrige Reaktionszeit der Steuerungsebene, Multi-Mandantenfähigkeit und automatische und dynamische Netzkonfiguration. In Kombination mit der Herausforderung einen robusten Betrieb zu garantieren, werden neuartige Designs benötigt die garantierte niedrige Reaktionszeit zu gewährleisten und zugleich die damit verbundene Bereitstellungskomplexität zu minimieren. Der Einfluss von verteilten Netzsteuerungsebene auf die resultierende Netzwerkleistung ist in der vorhandenenen Literatur wenig untersucht. Der Einsatz von SDN in industriellen Szenarien bedarf aber der Berücksichtigung des Problems der Zuverlässigkeit der Steuerebene. Für den Einsatz in Produktionsumgebungen wird dementsprechend der Nachweis einer robusten logisch-zentralisierten industriellen Steuerungsebene unbedingt erforderlich.

Wir postulieren fortan die Verwendung einer hochverfügbaren und ausfallsicheren SDN Controllerlösung als Wegbereiter für Ansteuerung und Betrieb der zukünftigen Industrienetze. Dementsprechend untersuchen wir die Eigenschaften der Verfügbarkeit, Zuverlässigkeit und Reaktionszeit der bereits vorhandenen konsensusbasierten Lösungen. Um die geringen Antwortzeiten und eine hohe Skalierbarkeit der verteilten Steuerungsebene zu garantieren, führen wir neue Mechanismen für effiziente Behand-

lung der Zustandsaktualisierungen in der verteilten Steuerungsebene ein. Außerdem definieren wir Mechanismen zur Toleranz semantischer Fehler in Controller-Replikaten, unabhängig von Grundursache des Fehlers, z.B. aufgrund von Software- und Hardware-Störungen, bösartiger Übernahmen der einzelnen Controller-Replikaten oder inkonsistenter Controller-Zustände. Wir stellen einen neuartigen automatisierten Bootstrapping-Ansatz vor, um die Bereitstellung einer zuverlässigen verteilten SDN Steuerungsebene in In-Band Netzen zu vereinfachen. Unsere Designs werden analytisch und empirisch validiert.

Kurz zusammengefasst adressiert unsere Arbeit die folgenden Punkte:

- *Auswertung und Härtung der SDN Steuerungssebene*: Wir definieren neue Methoden zur Sicherstellung der analytischen Garantien für Verfügbarkeit und Reaktionszeit der verteilten SDN Steuerungsebene. In der Leistungsanalyse setzen wir Steady-State und transitive Methoden basierend auf SANs ein. Wir werten zudem die Eckfälle aus, die sich negativ auf die Korrektheit der bestehenden Designs auswirken. Insbesondere werden Szenarien der Leaderoszillation und der erfolglosen Leaderwahl untersucht und reproduziert. Um diese zu lindern, entkoppeln wir das zugrunde liegende Fehlererkennungsverfahren von Konsensmechanismen des SDN Controllers.
- *Design einer skalierbaren fehlertoleranten verteilten Steuerungsebene*: Wir schlagen mehrere Entwürfe zur Realisierung eines Multi-Controller SDNs vor, die zugleich ein *Fail-Stop* -tolerantes, sowie einen skalierbaren Betrieb ermöglichen. Wir definieren ein adaptives Zustandsreplikationsmodell, dass jederzeit einen *ausreichenden* Konsistenzgrad für die ausgeführten SDN Anwendungen bereitstellt. Dabei werden die Divergenz der Zustandsaktualisierungen in Replikaten und damit die Worst-Case Leistung der ausgeführten Anwendung (z.B. Routing, Load Balancing usw.) eingeschränkt.
- *Design von Mechanismen zur Unterstützung eines zuverlässigen Betriebs einer verteilten Steuerungsebene*: Um eine korrekte Behandlung von Fehlern die auf byzantinischen Ereignissen beruhen zu gewährleisten, schlagen wir neuartige Mechanismen vor, die einen transparenten Systemübergang vom fehlerhaften in den stabilen Zustand garantieren. Somit wird ein korrekter Betrieb sichergestellt auch wenn bestimmte Controller-Replikaten unzuverlässige Ausgaben berechnen. In diesem Zusammenhang werden neuartige P4-basierte Mechanismen in der Datenebene eingesetzt um den Overhead der Replikation der Controller-Instanzen zu minimieren.
- *Automatisiertes Bootstrapping der hochverfügbaren und zuverlässigen verteilten Steuerungsebene*: Wir schlagen zwei neuartige Bootstrapping-Schemata vor, um ein komplexes verteiltes System bestehend aus einer beliebigen Anzahl von Controller-Replikaten zu initialisieren. Die In-Band-Steuerungsebene wird somit mit Verfügbarkeitsgarantien gebootet - d.h. sie wird automatisch gegen einzelne Datenebene- und Controller-Ausfälle geschützt.

Die überwiegende Mehrheit der in dieser Arbeit vorgeschlagenen Designs werden unter der Annahme eines industriellen Netzwerks ausgewertet, d.h. die Evaluierung der KPIs berücksichtigt die jeweiligen typischen Topologien und Parameterkonfigurationen. Die Vorteile eingeführter Designs sind jedoch auf andere Domänen transferierbar, z.B. auf Rechenzentrum- und Campus-Netze.

Acknowledgment

I am extremely grateful to my advisor, Prof. Dr.-Ing. Wolfgang Kellerer, for his patience and guidance during my doctoral program. Prof. Kellerer introduced me to the topic of SDN during my master's studies, and later continued to entrust me with his time, ideas, and collaboration, which both resulted in a deeper connection to the industry and ultimately the research results presented here. His open-mindedness allowed me to pursue exciting concepts and problems that went beyond the original research scope of the thesis. I am thankful for the provided lab equipment resources, as well as for his exposure of my intermediate results to the students and the chair staff, which ultimately resulted in numerous productive collaborations.

I greatly enjoyed my time at TUM and am thankful for the many friendships made during this time. I express my deepest gratitude to the current and alumni members of the 'wired' part of the chair for being close collaborators and most constructive critics of my work during the past four years: Dr.-Ing. Carmen Mas Machuca, Dr.-Ing. Petra Vizarreta Paz, Dr.-Ing. Amaury Van Bemten, Nemanja Deric, Amir Varastehhajipour, Dr.-Ing. Andreas Blenk, Dr.-Ing. Arsany Basta, Dr.-Ing. Raphael Durner, Dr.-Ing. Jochen Guck, Dr.-Ing. Mu He, Patrick Krämer, and Hasanin Harkous. The many retreats, social events, running competitions and food breaks would certainly not feel as complete without the 'wireless' half of the team, hence my sincere thanks to: Dr.-Ing. Murat Gürsu, Dr.-Ing. Mikhail Vilgelm, Onur Ayan, Arled Papa, Samuele Zoppi, and Serkut Ayvasik.

I benefited immensely from my association with Siemens Technology. In parallel to my doctoral obligations, Siemens offered me the environment to pursue my research in multiple company-internal and international research projects, thus providing the ground for establishing invaluable connections and sparking innovative ideas. I would especially like to thank Dr. Johannes Riedl for offering me the opportunity to pursue the doctoral degree in parallel to my scientific obligations at Siemens and for incorporating the research activities in my daily tasks, thus resulting in numerous publications and patents. Furthermore, I thank him for being my mentor during the whole duration of the program. Johannes taught me the importance for research projects to have real-world impact and has provided me with many opportunities to present and position my topics within Digital Industries and Mobility sectors within Siemens. I would also like to thank my team members Vivek Kulkarni, Dr.-Ing. Andreas Zirkler, Dr.-Ing. Matthias Scheffel, Dr.-Ing. Amine Houyou, Reinhard Frank, Hans-Peter Huth, Florian Zeiger, Dr. Joachim W. Walewski and Frimpong Ansah for the many technical discussions and reviews of my publication submissions.

I would like to pay my special regards to the students I had the honor of mentoring: Endri Goshi, Cristian Bermudez Serna, Sean Rohringer, Mirza Avdic, Valeh Sabziyev, Mantosh Kumar and Maheen Iqbal, who all assisted in achieving the outcome of this dissertation.

I would also like to express my gratitude to Prof. Dr.-Ing. Hermann de Meer and Prof. Sebastian Steinhorst for serving on my examination committee and for providing valuable feedback to the final version of this dissertation.

Finally, I would like to thank Ilona Fuchs for her support and encouragement. Above all, I would like to thank my parents, Rahima and Nevzet and my sister, Belma, for their enduring love and support. I dedicate this dissertation to them.

This dissertation was supported by the European Commission's Horizon 2020 research and innovation program under grant agreement number 671648 VirtuWind, grant agreement number 780315 SEMIoTICS, and in part under grant agreement number 647158 FlexNets (by the European Research Council).

To my parents and my sister.

Contents

Acronyms v

1 Introduction 1
- 1.1 Challenges and Thesis Contributions . . . 2
  - 1.1.1 Challenge I - Fault Tolerance Concerns . . . 2
  - 1.1.2 Challenge II - Tolerating Byzantine Faults . . . 3
  - 1.1.3 Challenge III - Response Time Concerns . . . 4
  - 1.1.4 Challenge IV - Deployment Complexity of Reliable In-Band Industrial Networks 5
- 1.2 Thesis Overview . . . 6

2 Motivation for a Highly Available and Reliable Network Control Plane 9
- 2.1 Target Systems and Use Cases . . . 9
  - 2.1.1 Industrial SDN Control Plane . . . 10
  - 2.1.2 802.1 Time-Sensitive Networking: Logically Centralized Network Functions 10
  - 2.1.3 Reliability of Industrial Control Systems . . . 12
- 2.2 KPIs and Related Challenges . . . 13
  - 2.2.1 Fault Tolerance and Correctness Property . . . 13
  - 2.2.2 Response Time Property . . . 13
  - 2.2.3 Configuration and Bootstrapping Complexity . . . 15

3 Enabling Fault Tolerance in Fail-Stop Control Plane 17
- 3.1 Cold- and Hot-Standby Redundancy in SDN Control Plane . . . 18
- 3.2 On Strong Consistency and Consensus . . . 19
  - 3.2.1 Strong Consistency Model . . . 19
  - 3.2.2 Consensus on Message Updates . . . 19
  - 3.2.3 Assumptions on the Industrial Network . . . 20
  - 3.2.4 Related Work - On Availability Estimation of a Distributed SDN Control Plane 21
  - 3.2.5 Raft Working Model . . . 22
- 3.3 Modeling the Availability of a Raft-enabled SDN Control Plane . . . 23
  - 3.3.1 Overview of Stochastic Activity Networks (SANs) . . . 23
  - 3.3.2 SAN Model Representations . . . 25
  - 3.3.3 Fast Recovery Mechanism for Bundles and Processes . . . 28

3.3.4 Evaluation and Results: Raft Cluster Availability 29
3.4 Supporting Partially-Observed Network Partitions: Decoupling Consensus from Event Detection 30
3.4.1 Related Work - On Correctness Issues with Raft 31
3.4.2 Problematic Scenario: Oscillating Leadership 31
3.4.3 Decoupling Consensus from Failure and Recovery Detection 33
3.4.4 Introducing the Event Agreement Component 34
3.4.5 Evaluation and Results: Validation of Correct Raft Operation with Network Partitions 36
3.5 Chapter Summary and Outlook 36

4 On Deterministic Response Time in Fail-Stop Control Plane 39
4.1 Analytic Guarantees for Raft's Response Time 39
4.1.1 Related Work - On Predictable Response Time of Cluster Operations 40
4.1.2 End-to-End Transaction Commit Model 41
4.1.3 Model parametrization using a Raft experiment 44
4.1.4 Evaluation and Results: Response Time Analysis 45
4.2 Upper Time Bounds on Distributed Failure Detection 49
4.2.1 Related Work - On Deterministic Convergence Time in Face of Network Partitions 50
4.2.2 Reference Model of Decoupled Failure Detection 51
4.2.3 Modeling Dissemination and Signaling Delays 52
4.2.4 Modeling Failure Detectors Delay 53
4.2.5 Modeling Event Agreement Delay 55
4.2.6 Evaluation and Results: Convergence Time of the Distributed Failure Detection 56
4.3 Chapter Summary and Outlook 59

5 Lowering Response Time in Fail-Stop Control Plane 61
5.1 Background on Different Consistency Models 61
5.2 Related Work - On Impact of Applied Consistency Model on SDN Performance . . . 63
5.3 Adaptive Consistency: Relaxing the Consistency Requirement 64
5.3.1 Design Overview 64
5.3.2 Relaxation of the Consistency Requirement 65
5.3.3 On Use of CRDTs for State Modeling 66
5.3.4 Performance Inspection Block 67
5.3.5 An Exemplary Inefficiency Metric for an Online SDN-LB 69
5.3.6 Online Consistency Adaptation Block 70
5.3.7 State Synchronization Strategies 71
5.4 Evaluation and Results: Comparison of AC, EC and SC in Distributed SDN Control Plane 72
5.4.1 Response Time Impact of Consistency Model Selection 75
5.4.2 Correctness of Application Decision Making 76

5.4.3 Overhead of State Distribution 78
5.5 Chapter Summary and Outlook 79

6 Enabling Fault Tolerance in Byzantine Fault Tolerant Control Plane 81
6.1 Related Work - On Enabling BFT in Context of SDN 82
6.2 MORPH: A Design Enabling Byzantine Fault Tolerant Operation 83
6.2.1 Terminology 83
6.2.2 System assumptions 83
6.2.3 Overview of Goals of MORPH 83
6.2.4 Architecture for Tolerating Byzantine Failures 84
6.2.5 Distinguishing Application Types 85
6.2.6 Necessity of Controller-to-Controller Synchronization 86
6.3 Detailed MORPH Design 87
6.3.1 Algorithm Specification 87
6.3.2 Optimized Controller-Switch Assignment 90
6.3.3 Communication and Computation Complexity 94
6.4 Evaluation and Results 95
6.4.1 Evaluation Scenario 95
6.4.2 Overhead minimization by successful failure detection 96
6.4.3 Impact of SDN Application Statefulness 98
6.4.4 Impact of proactive propagation of switch configurations 100
6.4.5 Controller-Switch Reassignment Time 101
6.5 Chapter Summary 101

7 Lowering Response Time and Control Plane Overhead of the BFT Control Plane 103
7.1 Control Plane Partitioning into Multiple Agreement Groups 104
7.1.1 Related Work - On Sequence and Value Agreement in BFT Designs 105
7.1.2 BFT State Synchronization Protocols 105
7.1.3 Enhancing MORPH's Controller-to-Switch Assignment Model 110
7.1.4 Evaluation and Results 112
7.2 Control Plane Acceleration by Offloading Tasks to the Data Plane 116
7.2.1 Related Work - On Data Plane-Accelerated BFT 116
7.2.2 P4BFT: Enhancing the MORPH model 117
7.2.3 Identification of Optimal Processing Nodes 118
7.2.4 P4 Switch and `REASSIGNER` Control Flow 120
7.2.5 P4 Tables Design 120
7.2.6 Evaluation & Results 121
7.3 Chapter Summary and Outlook 125

8 On Deployment Complexity of Distributed Control Plane 127
8.1 Motivation for In-Band Bootstrapping of a Reliable Distributed Control Plane 128
8.2 Related Work - On In-Band Bootstrapping of Distributed Control Plane 129

8.3 Background and Terminology 131
8.3.1 General Model 131
8.3.2 Secure and Standalone Modes 131
8.3.3 In-band Mode 132
8.4 HSS - (R)STP-based Automated Distributed Control Plane Bootstrapping 132
8.4.1 Phase 1: Network Startup and Initial Address Distribution 133
8.4.2 Phase 2: Enabling a Functional and Resilient Control Plane 134
8.4.3 Phase 3: Dynamic Network Extensions 135
8.4.4 Control / Data Plane Failure Handling 136
8.5 HHS - Hop-by-Hop Iterative Automated Distributed Control Plane Bootstrapping . . 137
8.5.1 Phase 1: Network Startup and Initial Address Distribution 138
8.5.2 Phase 2 - Enabling a Functional and Resilient Control Plane 139
8.5.3 Phase 2b: Enabling a Resilient Control Plane 139
8.5.4 Phase 3: Dynamic Network Extensions 139
8.6 Selected Design and Evaluation Aspects 140
8.6.1 Flow Table Occupancy 140
8.6.2 (R)STP Timer Parametrization 140
8.6.3 Network Topology Discovery 141
8.6.4 Controller Placement Discovery 141
8.6.5 Overhead of Controller Clustering 142
8.6.6 Coping With Broadcast Storms in HHS 142
8.7 Evaluation and Results 142
8.7.1 Evaluation Methodology 142
8.7.2 Bootstrapping Convergence Time 143
8.7.3 Network Extension Time 146
8.7.4 Flow Table Occupancy 146
8.8 Chapter Summary and Outlook 147

9 Conclusion **149**
9.1 Thesis Summary 149
9.2 Concluding Remarks 150

Bibliography **151**

List of Figures **175**

List of Tables **179**

Acronyms

ϕ-FD ϕ-based Failure Detector 50

(R)STP (Rapid) Spanning Tree Protocol 127

A&E Agreement and Execution 103

AC Adaptive Consistency 61

ACK Acknowledgment 142

AMD Advanced Micro Devices 49

API Application Programming Interface 2

ARP Address Resolution Protocol 132

BEB-D Best Effort Broadcast Dissemination 36, 37

BFT Byzantine Fault Tolerance 3

C2C Controller-To-Controller 4

C2S Controller-To-Switch 4

CAP Consistency, Availability, Partition Tolerance 20

CAPEX Capital Expenses 127

CDF Cumulative Distribution Function 54

CL Consistency Level 62, 63

CLI2C Client-to-Controller 20

CLI2S Client-to-Switch 94

CNC Centralized Network Configuration 12

CPU Central Processing Unit 12

CRDT Conflict-Free Replicated Data Types 64

CSPF Constrained Shortest Path Forwarding 112

CTMC Continous Time Markov Chain 23

CUC Centralized User Configuration 12

DHCP Dynamic Host Configuration Protocol 129

DHCPv6 Dynamic Host Configuration Protocol Version 6 148

DNS Domain Name System 132

DTMC Discrete Time Markov Chain 25

E2E End-To-End v

EC Eventual Consistency 61

ECDF Empirical Distribution Function 45

EMI Electromagnetic Interference 15

FCFS First-Come-First-Serve 133

FIFO First-In, First-Out 71

FTO Flow Table Occupancy 140

GADAG Generalized Almost Directed Acyclic Graph 11

GBCT Global Bootstrapping Convergence Time 143

GCL Gate Control List 10

gRPC gRPC Remote Procedure Calls 122

HHS Hop-by-Hop Scheme 130

HSS Hybrid Switch Scheme 130

HTTP Hypertext Transfer Protocol 122

IBC In-Band Network Control 5, 127

IEC International Electrotechnical Commission 12

IEEE Institute of Electrical and Electronics Engineers v

ILP Integer Linear Program 81

IO Input / Output 12

IoT Internet of Things 12

IP Internet Protocol 6

IPv4 Internet Protocol v4 131

IPv6 Internet Protocol v6 141

ISO International Organization for Standardization 9

ISP Internet Service Provider 74

KPI Key Performance Indicator vi

L-EA List-based Event Agreement 34

L2 Layer 2 14

LLDP Link Layer Discovery Protocol 134

LLDPDU LLDP Data Unit 141

MAC Message Authentication Code 83, 131

M-EA Matrix-based Event Agreement 34

MAC Media Access Control 33

MPBFT Modified PBFT 106

MRT Maximally Redundant Tree 11

N.E.D. Negative Exponential Distribution 25

NBI Northbound Interface 84

NETCONF Network Configuration Protocol 73

NEXT Network Extension 134

NME Network Management Entity 12

NMS Network Management System 14

NPU Network Processing Unit 126

OBFT Opportunistic A Posteriori BFT 105

OCA Online Consistency Adaptation Block 65

ODL OpenDaylight 2

OF OpenFlow 6

ONOS Open Network Operating System 2

OOBC Out-Of-Band Network Control 5, 127

OSGi Open Services Gateway initiative 25

OVS Open vSwitch 44

OVSDB Open vSwitch Database Protocol 134

P4BFT P4-Enabled BFT Control Plane 103

PBFT Practical BFT 104

PC Personal Computer 75

PI Performance Inspection Block 65

PID Proportional, Integral, Derivative Control 65

PKI Public Key Infrastructure 131

PLC Programmable Logic Controller 9

PN Petri Net 24

PN-Counter Positive-Negative Counter 63

PR-S Ping-Reply Signaling 53

QoS Quality of Service 2

RAM Random Access Memory 75

REST Representational State Transfer 72

RF Radio Frequency 15

RPC Remote Procedure Call 20

RRG-D Round Robin Gossip Dissemination 36, 37

RSM Replicated State Machine 13

RSTP Rapid Spanning Tree Protocol 141

S2C Switch-to-Controller 15

SAN Stochastic Activity Network vi

SBFT Serialized A Priori BFT 106

SBI Southbound Interface 21

SC Strong Consistency 4

SDA State Dependent Application 85

SDK Software Development Kit 122

SDN Software Defined Networking v, 1

SDN-LB SDN-based Application Load Balancer 66

SGX Intel Software Guard Extensions 85

SHA-256 Secure Hash Algorithm 113

SIA State Independent Application 85

SLA Service Level Agreement 10

SLAAC Stateless Address Autoconfiguration 148

SPOF Single Point Of Failure 2

SSH Secure Shell 131

STP Spanning Tree Protocol 141

TCP Transmission Control Protocol 95

TCP SYN TCP Synchronize Message 142

TEXT Time To Extend 143

T-FD Timeout-based Failure Detector 53

TAS Time-Aware Shaper 1

TOR Time Of Request 108

TSN Time-Sensitive Networking v

UH-S Uni-Directional Heartbeat Signaling 53

VLAN Virtual LAN 10

VM Virtual Machine 29

VNF Virtual Network Function 1

vPLC Virtualized Programmable Logic Controller 12

WAN Wide Area Network 127

WD Watchdog Mechanism 44

YANG Yet Another Next Generation 72

Chapter 1

Introduction

Software Defined Networking (SDN) is a recent paradigm in control and management of communication networks [64, 95, 106]. In contrast to a fully distributed network control model, where network logic is shared across multiple autonomous network devices (e.g., network switches, routers and Virtual Network Functions (VNFs)), SDN centralizes the decision-making logic in a single configuration entity - the SDN controller. Thus, with SDN, the network logic (e.g., traffic engineering [75], virtual network embedding [31, 43], load balancing [2, 113] etc.) and the network knowledge state (e.g., resource mappings, topology graphs, forwarding tables etc.) are present in the controller instance responsible for its administrative domain. The data is exposed to higher-layer applications (i.e., data consumers) using a set of *northbound* interfaces, hence enabling a simplified access to the network logic and knowledge state, compared to a fully distributed network control plane.

The advantages of the SDN architecture have recently been transferred to industrial networks (e.g., in automotive, energy, factory automation, healthcare domains), providing for significant advances w.r.t., network visibility [124, 165], intrusion detection and prevention [13, 152] and dynamic routing and resource reservation [74, 75]. Thus, the overhead of configuration engineering and the configuration staticity of Industrial Ethernet networks [63, 190] is superseded by the dynamicity of network control and management introduced by SDN [9]. In fact, IEEE's TSN group has recently specified and standardized a set of Standard Ethernet extensions to support Industry 4.0 use cases with requirements on dynamic point-to-multipoint stream establishment, low and bounded forwarding latency and high availability of established streams [204, 213, 67, 92, 93, 224]. The corresponding amendments to (majorly) the IEEE 802.1Q standard [93] (Bridges and Bridged Networks) have adopted and assume the centralized approach to network control and management. The centralized SDN approach is preferred in scenarios where global network visibility and serialized decision-making is necessary in order to compute and deploy complex configurations for solving multi-constrained problems, e.g., for bandwidth and delay-constrained multicast stream-establishment assuming Time-Aware Shaper (TAS) [30, 84, 147, 172].

An SDN approach assumes logical centralization of the networking knowledge. This necessarily poses a number of dependability-related concerns: (i) the impact of single-controller deployment on the resulting network service availability; (ii) the reliability of single-controller decisions; (iii) the

imposed response time drawback by the centralization of network control Application Programming Interfaces (APIs); and (iv) the incurred overhead of deployment complexity.

We next discuss these concerns in more detail and refer to the approaches of *distributed SDN control plane* that aim to address a subset of the concerns. We additionally summarize the remaining challenges specific to industrial SDN deployment. Finally, we briefly discuss the contributions made by this thesis, encompassing analysis, design and optimization of the *distributed SDN control plane*.

1.1 Challenges and Thesis Contributions

1.1.1 Challenge I - Fault Tolerance Concerns

Centralized network control introduces the issue of a Single Point Of Failure (SPOF). A controller failure in an SPOF deployment necessarily leads to loss of a network control and management interface to forwarding devices. This issue is of fundamental importance in critical industrial networks that often must achieve extremely high system dependability [134].

Various approaches to enabling a redundant operation of the SDN controllers were proposed to alleviate the SPOF issue [39, 98, 100, 115, 123, 171, 183]. Most notably, controller *clustering* designs [39, 123] enable a *quorum*-like operation of the SDN controller cluster, where after reaching *consensus*, each knowledge-base update in one of the controllers is propagated across the cluster members, so to enable the consistency in ordering and value changes of state updates. In a leader-based design, whenever the *leader* controller instance fails, consistent backup controller replicas autonomously decide on the new *leader*. Hence, eventually the SDN control plane stabilizes.

For dependability, certification and Quality of Service (QoS) guarantees, critical infrastructure providers require latency and availability guarantees on partaken reconfigurations [3, 88, 224, 193]. As discussed above, SDN can enable the necessary robustness against *Fail-Stop* failures by controller clustering and controller state replication. The replication, however, incurs additional performance overhead. Indeed, consensus systems typically assume a single-leader operation (or a centralized *serializer*), thus imposing a non-negligible system fail-over time after leader (or *serializer*) failure. The existing literature generally neglects the impact of failure occurrences on control plane's response time. In particular, no guarantees on control plane performance in failure case are given by present designs. Characterization of expected response time is hence typically based on limited experimental validations for fixed control plane and topology parametrizations [181, 182].

Our contributions in this context are two-fold.

Contribution 1 - Analytic models for availability and response time evaluation of a distributed SDN control plane: We propose the usage of SANs [44] for modeling and numerical evaluation of distributed SDN controller clusters [3]. Specifically, Chapter 3 and Chapter 4 describe a framework for exact modeling of the distributed consensus algorithm Raft [146] used in production-ready SDN controller platforms OpenDaylight (ODL) and Open Network Operating System (ONOS). The developed models are applicable to other leader-based consensus algorithms as well (e.g., Multi-Paxos [218]).

The solutions to the introduced SAN-based models allow for response-time and availability characterization of arbitrary cluster parametrizations. Furthermore, a failure injection and controller recovery models are proposed for evaluation of the two metrics under the effect of an arbitrary number of correlated controller failures. Using transient and steady-state analysis, we provide analytical guarantees on the event handling response times and long-term availability of an arbitrary SDN control plane configuration, respectively. Thus, with the proposed approach, evaluation and optimization of optimal cluster configuration for an industrial network becomes possible without costly hardware setups for experimental evaluation and lengthy simulation runs.

Contribution 2 - Correctness guarantees in distributed SDN control plane in the face of network partitions: In the face of network partitions, a distributed system becomes unable to progress its state in each of the active replicas. To agree on the existence of potential partitions, the consensus abstractions hence require a deployment of a minimum of $2 \cdot F + 1$ controller instances in order to tolerate a maximum of F instance failures. However, in the case of network partitions with partial reachability, where individual instances are mutually reachable but an any-to-any connectivity matrix cannot be established, the consensus protocols may encounter an unstable state where no leader can be identified due to a *leader oscillation* issue. In Section 3.4 we present this issue and introduce, validate and evaluate a design to decoupling the underlying failure detection process from state consensus protocol capable of stabilizing the consensus in the face of network partitions. Additionally, Section 4.2 provides an analytic framework for computation of the corresponding convergence times necessary to converge the distributed failure detection process in each active controller replica.

1.1.2 Challenge II - Tolerating Byzantine Faults

Single-leader distributed control plane designs are susceptible to *Byzantine* faults, which directly impact the system reliability. Byzantine faults result in failures where imperfect information is available on whether a component has indeed failed [54]. They can, for example, be caused by a malicious adversary in possession of any controller replica [115, 128]. Additionally, they can have origin in internal semantic or logic faults in the replica, e.g., caused by the aging of controller software [18, 189] or state inconsistencies [113]. Byzantine faults may eventually result in a faulty replica computing an output to a client request that deviates from expected correct output and thus potentially destabilizes the overall system.

Reliability of decisions of a control entity is of utmost importance for operation of critical infrastructure [179], e.g., in automotive or avionics networks where consistent and correct configuration of network flows is necessary to enable the connectivity of fail-safe applications deployed in end-points. To this end, Byzantine Fault Tolerance (BFT) has been thoroughly investigated in the automotive [62, 103, 179] and train control [227] domains, especially with regard to control application execution. BFT in the networking and SDN context has, however, only recently started gaining attention in the research community [115, 128].

Contribution 3 - An adaptive design for efficient BFT SDN control plane: In Chapter 6, we introduce MORPH [1], a design capable of distinguishing Byzantine from Fail-Stop controller failures

at runtime. Following a Byzantine or Fail-Stop controller failure, MORPH autonomously adapts the cluster membership configuration, so to minimize the distributed control plane overhead. The proposed design achieves a performance improvement both for average- and worst-case system response time by optimally deducing the amount of active controllers at runtime. The QoS-constrained reassignment of Controller-To-Switch (C2S) relationships is formulated as an optimization problem, capable of online execution in large-scale control planes comprising at least up to 16 controller replicas. The time required to re-adapt the system configuration scales proportionally with the number of successfully detected faulted controllers. Average- and worst-case computational and communication overheads are significantly decreased compared to alternative designs [115, 128].

1.1.3 Challenge III - Response Time Concerns

Consensus algorithms, such as Raft and Paxos [87, 110], guarantee the property of Strong Consistency (SC) of control plane state. With SC, each system state update is propagated among all active controller replicas and is confirmed by the majority of the cluster members, prior to being marked as committed (i.e., and eventually executed). Putting the issue of single-leader election discussed in Section 1.1.2 aside, the response time drawbacks when deploying consensus and hence SC are multi-fold:

1. The replicated control plane runs at the speed of the slowest replica in the cluster majority - that is, the replica taking the longest time to apply its local update will block the leader in its execution. This in return results in a considerable system throughput decrease, compared to single-controller execution.

2. Distributing the control plane updates to multiple replicas necessitates consensus. During the synchronization period, the control plane does not progress its state and blocks indefinitely until the agreement on update order is reached by the majority of controller replicas.

3. In the face of a leader failure and during a leader re-election period, the system does not progress its state until a majority vote has been decided on the next term leader. Hence, during the leader-failed period, the system is unavailable for serving any external user requests.

4. Convergence time after a leader failure is governed by the speed of the slowest controller in the fastest cluster majority. Leader election procedure initiates the selection of the new cluster leader and subsequent agreement on the new leader, only after a certain minimum timeout has expired. Hence, the control plane's long-term average unavailability is a function of the number of deployed replicas, as well as the network design and runtime factors, such as Controller-To-Controller (C2C) communication and processing delays.

Industrial networks explicitly define maximum response time requirements on E2E network stream establishment. We discuss these in Section 2.2.2 but summarize our contributions related to response time minimization here.

Contribution 4 - Response time minimization in Fail-Stop approaches using flexible consistency models: Section 5 investigates the impact of a particular consistency model's deployment on

the overall control plane performance. To this end, we evaluate the state-of-the-art *eventual* and *strong* consistency models and compare them to our *adaptive consistency* approach [2, 10]. *Adaptive consistency* ensures that certain invariants on state synchronization divergence hold among all controller replicas at all times. It provides the state divergence guarantees under efficient operation by means of runtime adaptation of the active *consistency levels*. A consistency level property is individually assigned to every resource state accessed by an SDN application (e.g., a path finder or load-balancer) and is adapted based on the the inefficiencies resulting from operations using stale state. The proposed method enables the design of a scalable SDN control plane relying on state replication, since, in contrast to a strong consistency setup, the majority of state updates execute as non-blocking, eventually consistent operations.

Contribution 5 - Minimization of response time of a BFT distributed control plane using dynamic agreement groups: In Section 7.1 we introduce a design for minimizing the overhead of command ordering in a BFT deployment of the distributed SDN control plane. In particular, we propose and implement two agreement-based and an opportunistic BFT protocol for request sequencing and state synchronization in Byzantine scenarios, and analyze their overheads in experimental environments [7]. The proposed protocol designs offer a considerably decreased response time compared to the sequencing-based approaches presented in existing literature.

Contribution 6 - Minimization of incurred control plane load of a BFT distributed control plane using control plane task-offloading: Section 7.2 extends the previous contribution by minimizing the overall control plane load generated in a BFT setup. We introduce an approach relying on programmable data plane primitives, i.e., the P4 [46] data plane programming abstraction, to intercept C2S control plane flows and agree on the correct control message prior to the packets' delivery at the destination switch's software control plane. The offloading thus leads to a decrease of resulting incurred data plane overhead without loss of correctness of the BFT process [46, 11].

1.1.4 Challenge IV - Deployment Complexity of Reliable In-Band Industrial Networks

State-of-the-art SDN deployments in data-centers typically assume an Out-Of-Band Network Control (OOBC) control plane deployment, where controller replicas manage the forwarding plane using a physically isolated, dedicated management network. A cost-effective multi-controller control plane in typical industrial scenarios will, however, entail an In-Band Network Control (IBC) network setup. With in-band control, data plane and control plane flows must share the same forwarding substrate. However, a a reliable IBC deployment must consider two requirements.

First, an IBC control plane must fulfill the stringent high-availability and reliability properties: i.e., the IBC control flows must be isolated from data plane traffic, and the impact of data plane failures on control plane stability must be minimized. Second, a distributed control plane imposes the following invariant: No configuration updates may be applied in the network, as long as all controllers have not agreed to the planned configuration change.

These two requirements pose the fundamental challenge: *How to bootstrap a reliable distributed SDN control plane, and accordingly enable the C2C connectivity if, by definition, no switch configurations may execute without a prior C2C consensus*?

Contribution 7 - Bootstrapping schemes for automated deployment of reliable distributed IBC SDNs control planes: In Chapter 8 we answer this question and design and evaluate two IBC bootstrapping schemes that autonomously bootstrap a multi-controller SDN with a resilient control plane and automatic Internet Protocol (IP) address and OpenFlow (OF) controller list provisioning to the switches. We validate the practicability of both approaches and quantify their trade-offs in realistic industrial network topologies with up to 64 OF switches deployed in a single broadcast domain. The schemes are evaluated with regard to their computation complexity, legacy protocol requirement, convergence time, flow table occupancy, and network extension time and have been successfully demonstrated in an industrial network with critical fail-safe requirements.

1.2 Thesis Overview

The organization of thesis is visualized in Fig. 1.1.

Chapter 2 discusses the exemplary industrial control plane applications requiring fault-tolerant SDN operation and thus details the KPIs of relevance for the work conducted in this thesis. Additionally, it contains the summary of contributions related to each of the presented KPIs.

Chapter 3 details the state-of-the-art distributed control plane designs used to tolerate Fail-Stop failures, prevalent in existing major controller platforms. It then proceeds to present the analytical models that leverage SANs for estimation of resulting long-term control plane availability. It discusses a number of corner-case issues related to correctness of the distributed consensus that arise after occurrence of network partitions and presents and validates a decoupled interaction between the distributed consensus and underlying failure detection to resolve these issues.

Chapter 4 presents and solves the SAN models for determining the upper bounds on control plane response time after occurrence of replica failures. Next, the chapter investigates the scalability and accuracy of the underlying discrete-time Markov process relevant for large-scale control plane deployments. The remainder of the chapter introduces an analytic formulation to worst-case convergence time of the orthogonal failure detection process. The derived worst-case directly influences the unavailability incurred by the distributed leader election procedure.

Chapter 5 investigates the impact of the deployed consistency model on scalability and correctness metrics of the distributed SDN control plane. It proposes, validates and evaluates a novel *adaptive consistency* approach. It empirically demonstrates how application of the adaptive model results in a higher request-handling throughput and a decreased response time time, compared to the strong consistency approach, while guaranteeing correctness semantics unavailable with eventual consistency semantics.

Chapter 6 describes a novel control plane design capable of tolerating both Fail-Stop and Byzantine failures. It furthermore optimally solves a C2S connection assignment problem by leveraging the

awareness of the source of failure (i.e., Fail-Stop or Byzantine-induced), thus minimizing the number of active controllers and the resulting controller and switch reconfiguration delays.

Chapter 7 investigates the optimizations for response time and packet load overhead minimization in BFT scenarios. It additionally investigates the benefits of offloading the procedure of control plane packet comparison, required for identification of Byzantine replicas, to carefully selected network switches, thus decreasing the resulting control plane network load.

Chapter 8 introduces two automated bootstrapping schemes for a multi-controller IBC distributed control plane resilient to link, switch, and controller failures. The proposed schemes enable fast bootstrapping of a robust industrial SDN with proactive redundancy and network extension support, while being interoperable with off-the-shelf OF switches.

Chapter 9 concludes the thesis.

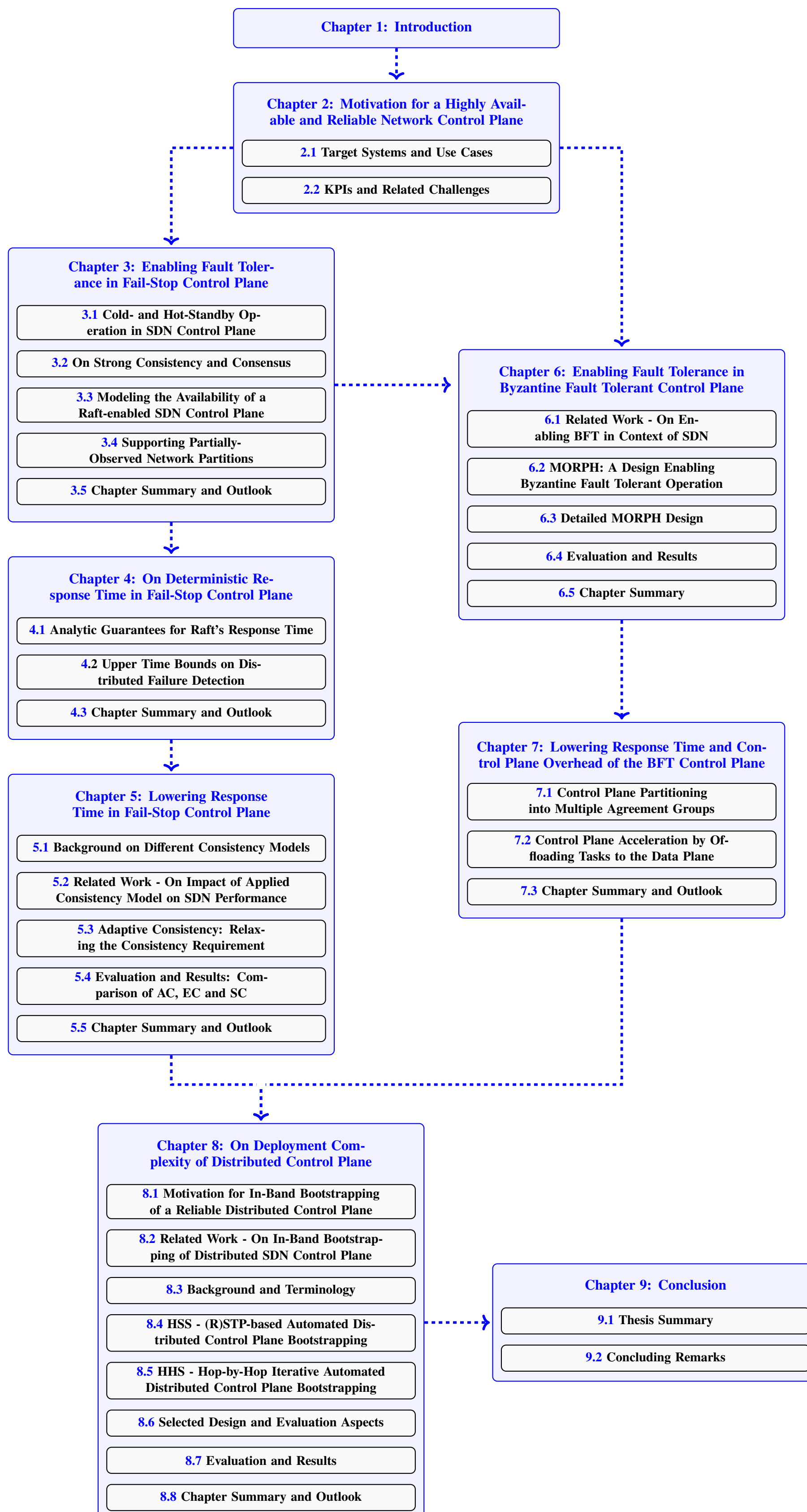


Figure 1.1: Structure of the thesis.

Chapter 2

Motivation for a Highly Available and Reliable Network Control Plane

International Organization for Standardization (ISO) 25000 [97] defines *reliability* as the degree to which the system performs a specified function under specified conditions for a specified period of time. *Availability*, on the other hand, is a sub-component of reliability and defines the degree to which the system under consideration is operational and accessible to use at a given point in time. Thus, an *available* system may be operational but performing in a manner incompatible with its expected correct state. Specifically in the context of SDN controller instances, we consider a controller *available* if, during the observed period, it is capable of delivering a response to client requests. In contrast, an active controller capable of providing a guarantee of delivering a *correct* output given an arbitrary input, is considered both available *and* reliable.

Industrial services often require a highly-available deployment [90], with a subset of mission-critical applications requiring tolerance against reliability faults [179, 227]. While a highly-available *and* reliable industrial control plane design may seem optimal in any industrial SDN, providing for reliable operation comes at a cost of increased computational and communication overhead.

Designs proposed in this thesis enable a highly-available and reliable operation of controller instances. Providing availability guarantees is achieved by tolerating *Fail-Stop* (also *Fail-Silent*, *Fail-Fast*) failures [49], i.e., types of failures that result in an eventual halt of the faulty instance. To this end, we rely on using consensus-based controller state replication. Reliability guarantees are provided by means of a BFT design that tolerates Byzantine faults. Prior to describing the approaches in detail we first clarify the exemplary industrial control plane applications requiring either type of fault tolerance.

2.1 Target Systems and Use Cases

The following section gives an overview of exemplary industrial control plane applications necessitating redundant operation to tolerate Fail-Stop and Byzantine faults. While the focus of the thesis is put on distributed SDN control plane, the presented approaches are applicable to other critical control functions that benefit from increased availability and reliability, including Programmable Logic Controllers (PLCs) and selected TSN functions.

2.1.1 Industrial SDN Control Plane

In critical infrastructure networks, such as in the utility [90, 119, 153, 173] and automotive [103, 179] domains, resilience of the communication network is an important criterion for adopting a disruptive network technology such as SDN. The unavailability of the SDN controller leads to loss of control and monitoring channels with the network devices, risking the auditing capabilities of the system and, potentially, a system instability.

An industrial SDN control plane may be deployed in different manners:

- *Focus on network service engineering*: The network is designed to operate in a fail-safe manner even after a controller failure. For example, the controller may be used as an engineering tool, which, following its failure, does not impact the ongoing operation of the (reliable) data plane. Thus, the SDN controller is deployed as a non-critical function, used for rollout of pre-planned initial switch configurations (e.g., forwarding rules, 802.1Qbv Gate Control List (GCL) schedules [216], Virtual LAN (VLAN) configurations etc.) and used at runtime only for e.g., passive data collection and logging, failure localization and optional traffic engineering.

- *Focus on online service admission*: SDN control plane supports the dynamic applications, e.g., industrial 5G and multi-tenancy scenarios that require dynamic provisioning of QoS-constrained flows and dynamic reservation for newly registered services, so to enable efficient utilization of network resources shared by multiple tenants. The control plane failures there lead to an unavailability of the flow admission service and thus a degraded user experience. Furthermore, unavailability of basic services may enact unplanned operational losses due to Service Level Agreement (SLA) invalidations.

Deployment of a distributed control plane is, however, beneficial in both scenarios, i.e., providing for highly-available network monitoring in the first case, and the transition to a backup resource reservation instance on failure of the main instance, in the latter case. The criticality and importance of timely fail-over motivates the differentiation of two controller redundancy models: the *hot-* and *cold-standby* controller redundancy (discussed in later Section 3.1).

2.1.2 802.1 Time-Sensitive Networking: Logically Centralized Network Functions

Time-Sensitive Networking (TSN) is a family of standards under development by the TSN task group of the IEEE 802.1 working group. TSN standardizes the extensions to Ethernet, targeting the support for time-sensitive (i.e., bounded latency) applications used in e.g., factory automation, process industry and automotive applications. In particular, the standards address the gap of inadequate time synchronization, per-flow (i.e., per *stream*) scheduling, traffic shaping and policing, frame preemption, dynamic path reservation, and per-stream fault-tolerance, in the existing Ethernet specifications.

In terms of the TSN control plane, IEEE proposes the path reservation protocol and bridge configuration models for centralized control plane in the 802.1Qca [92] and 802.1Qcc [93] amendments to the 802.1Q standard, respectively. In particular, [93] specifies distributed, centralized and hybrid

configuration models to managing the TSN mechanisms. All three approaches rely on a number of logical functions requiring highly-available and reliable operation for ensuring data plane correctness. In particular, the dynamic stream establishment using highly-available and reliable centralized admission control in a single and across multiple TSN domains, as well as a redundant clock source function for supporting time-aware shaping, policing and frame preemption must be supported by the resulting implementation. TSN standards, however, do not specify exact mechanisms and redundancy protocols used to realize these requirements.

In the remainder of the section, we discuss these individual requirements in more detail.

2.1.2.1 GADAG Computer and Grand Master Functions

Generalized Almost Directed Acyclic Graph (GADAG) Computer Redundancy: In order to provide for redundant point-to-multipoint streams, 802.1 TSN-enabled bridges compute Maximally Redundant Trees (MRTs) that stretch across all TSN domain forwarding elements. To ensure consistency of packet propagation along the computed trees, bridges compute these based on a unique GADAG, distributed to each bridge by the current GADAG Computer [207]. Upon reception of a new or updated GADAG, each TSN bridge computes the MRTs, in isolation of other bridges. The use of a GADAG computer hence solves the issue of inconsistent forwarding on the current topology.

To persist bridge operation upon failure of the current GADAG, a redundancy and synchronization mechanism must be made available, guaranteeing ordered GADAG computation in the GADAG Computer, and the eventual propagation of GADAG to all bridges in the topology. The centralization of GADAG computation, however, represents a SPOF and motivates the need for its redundant and strongly consistent deployment.

Grand Master Redundancy: IEEE 802.1AS-2011 defines a method to establish a clock hierarchy in order to synchronize the time in each device of the common TSN domain, relative to a common reference clock. To account for the data path delays, the proposed protocol measures the frame residence time within each bridge (comprising processing, queuing and ingress-to-egress port transmission delays), and the link latency of each hop. The calculated delays of all bridges are then referenced to the Grand Master clock with clock advertisement paths established using a clock spanning tree protocol. The Grand Master is elected based on the Best Master Clock Algorithm, taking clock accuracy, clock class, clock variance and priority values into account.

In case of a failure of an existing Grand Master, a secondary Grand Master replica must take over. We identify two main requirements related to establishing its redundant operation:

1. On current primary failure, a fail-over to a secondary Grand Master must take place. However, a maximum of one Grand Master source clock may be elected at any point in time.

2. The base standard specifies a fixed "announce timeout" interval. After exceeding this interval without updated clock announcements, secondary instances will take-over the role of new Grand Master. In the case of network partitions, however, coupling the failure detection with Clock Master operation may lead to "flapping" of the current Grand Master [201]. This problem

is related to any replicated components relying on leader election, and is discussed further in context of SDN controller replication in Section 3.4.

2.1.2.2 Centralized Network and User Control Functions

Analogue to the SDN controller operation, 802.1Qcc amendment [93] proposes two components responsible for network and end stations' configuration in a dynamic manner: (i) the Centralized Network Configuration (CNC) - a component that configures network resources on behalf of TSN applications (users); and (ii) the Centralized User Configuration (CUC) - a centralized entity that discovers end stations, retrieves end station capabilities and user requirements, and configures TSN features in end stations.

In International Electrotechnical Commission (IEC) / IEEE 60802 standard draft [224], both components are coupled in the single logical entity called Network Management Entity (NME). The NME is responsible for typical SDN controller functions, including the configuration of end-points' application parameters. Additionally, NME is capable of east-westbound communication to NME instances of neighboring TSN domains (i.e., other administrative domains), so to enable provisioning of inter-TSN-domain path computation and enforcement.

The requirements of a highly available and reliable NME are the same as in the generic SDN scenario presented in Section 2.1.1. The cross-domain stream establishment further motivate the need for reliable NME, due to the cross-domain interactions relying on their correct operation.

2.1.3 Reliability of Industrial Control Systems

Ubiquitous redundancy solutions are necessary in industrial field control and fault-tolerant Internet of Things (IoT) devices as well [208, 179]. There, PLCs used in open- or closed-loop control may be deployed with redundant highly-available Central Processing Units (CPUs) [219], so to support redundant hot-standby execution in case of primary instance failure. For example, Siemens's SIMATIC S7-400H [219] supports automatic event-driven synchronization of internal processing events between the master and backup CPUs so to enable an interrupt-free operation by the secondary CPU even if the master CPU fails. Similarly, the Input / Output (IO) devices used in sensor reading and actuation may be deployed or interconnected redundantly to the PLC CPUs. The CPUs are, however, connected by two multi-mode fiber-optic cables with maximum cable length of up to $10m$, or single-mode fiber-optic cables with a maximum cable length of up to $10km$, thus limiting the deployment of the approach in virtualized environments that rely on Virtualized Programmable Logic Controller (vPLC) execution [56].

Designs presented henceforth extend the event synchronization capability to more than two application replicas and support arbitrary distribution of physical / virtual network functions, while nonetheless guaranteeing the convergence time on deterministic detection of failed instances. Although we henceforth put focus on network control functions, the presented approaches may be applicable to the domain of distributed virtualized PLCs and fault-tolerant IoT and industrial edge devices as well [208].

2.2 KPIs and Related Challenges

Fail-Stop failures in the SDN control plane can be addressed by distributing the logical control plane among multiple virtualized or physical resources [39, 100, 106, 183]. In the case of a primary instance failure, a secondary instance may take-over the primary's role. However, the multiple instances' deployment comes with an overhead of an added response time, incurred by blocking on synchronization events and proximity of replicas, a prolonged convergence time after failures, and the deployment complexity. Similarly, Byzantine (i.e., reliability) faults are not tolerated by the designs addressing *Fail-Stop* failures only.

Based on the discussed industrial use cases, we next derive a set of capital system requirements specific to the industrial scenarios and summarize the contributions to those properties presented in this thesis.

2.2.1 Fault Tolerance and Correctness Property

In single-controller SDN scenarios, unavailability of the controller leads to loss of control and monitoring channels with the network devices and hence a system instability. The loss of network control may result in production and power outages in smart grid domain [90] or even life-threatening scenarios in dependable automotive domain [179]. An industrial system may be impacted with Byzantine and Fail-Stop faults, respectively impacting reliability and correctness of decision-making or only the availability of the control plane.

As a consequence of a Byzantine fault, the system may compute wrong decisions, while in the case of a Fail-Stop fault, the impacted instance halts and stops responding to client requests. Some SDN applications, e.g., load-balancing of non-critical client-server requests, collection of the data for monitoring purposes or statistics processing etc., may indeed tolerate reliability-related failures with collected statistics or decisions being incorrect or suboptimal [113]. In others, e.g., for time-critical TSN stream establishment, faulty network reconfigurations can impact the correctness of the overlaying mission-critical applications and thus the overall system dependability [224, 179].

Our contribution: To enable efficient designs for supporting both types of mission-criticality, we introduce the approaches for providing basic fault-tolerance for non-mission critical SDN applications in Chapter 3. For application domains requiring strict correctness of decision-making, in the face of Byzantine faults (sourced in e.g., inconsistencies or aging-related bugs [18, 189]) we present a Replicated State Machine (RSM)-based design of multi-controller execution in Chapter 6.

2.2.2 Response Time Property

Critical infrastructure providers often have stringent requirements on the experienced control plane delay. For example, the smart grid is a delay-sensitive infrastructure that requires techniques which identify and react to any abnormal communication network changes in a timely manner. If the detection and responses are not made promptly, the grid may become inefficient or even unstable and cause further catastrophic failures in the entire network [90]. Events in the grid may require rapid

reaction from the network controller - e.g., rerouting after power grid failures, expedited diagnostics and alarm handling [15].

Furthermore, Network Management Systems (NMSs) in the 5G context can require bounded configuration times when establishing on-demand network services [10]. Recent IEC / IEEE activities have motivated the support for dynamic network extensions in machine assemblies [213, 224]. These impose the requirement of Layer 2 (L2) multicast tree establishment with response time guarantees using a reliable network controller (i.e., a Raft-enabled distributed SDN controller [39, 123]). Thus, the response time of the distributed network control plane must be minimized to guarantee the upper bound value for tree computation and enforcement in under $500ms$ [224].

The total resulting response time comprises two delay sources: i) the time taken to process the client requests (e.g., route computation and enforcement) and ii) the time delay during control plane recovery after transient data plane / control plane failures during request processing. We next distinguish and discuss these components in more detail.

2.2.2.1 Handling Controller Requests

SDN controllers may host arbitrarily complex application logic, e.g., routing algorithms or algorithms for data analysis relying on large data sets. The worst-case response time, e.g., $< 500ms$ required by [224], must not be exceeded by the computation task of the responding controller instance independent of the task's complexity.

Additionally, the distribution of stateful controller replicas imposes significant synchronization overhead in terms of time delay necessary to exchange, apply state updates and maintain consistent state in each of the instances [181, 182]. Both the intrinsic computation time and the synchronization delay must hence be considered separately in the worst-case analysis of the system response time.

2.2.2.2 Convergence Time on Control Plane Reconfigurations

The worst-case response time must also consider the possibility of controller and network failures during idle time or request handling. For example, on occurrence of a controller failure or loss of communication link to the currently elected master instance, a backup replica may need to be elected a new master, and subsequently re-initiate failed client requests, thus increasing the operational per-request delay.

Similarly, network partitions and unreliable communication channels between particular replicas can cause the state-of-the-art failure detectors of off-the-shelf distributed controller distributions [39, 123] to miss meeting the response time demands. In this light, Zhang et al. [201] have demonstrated the issue of *oscillating leadership* when using distributed consensus for synchronization of replicated SDN controller instances.

Our contribution: Chapter 4 discusses the analytic methods for identifying deterministic response time of the distributed control plane capable of tolerating Fail-Stop failures. In particular, Section 3.3 and Section 4.1 propose the usage of SANs to model and bound the system response time of a

single-leader distributed control plane for a given parametrization, with support for transient controller failures. Section 4.2 continues with an analytic approach to bounding the worst-case failure detection time with support for network partitions (resulting from data plane failures). When requiring tolerance against Byzantine faults, however, we guarantee seamless fail-over to backup replicas by relying on replicated concurrent execution of controller tasks, leading to no interruptions at expense of a higher computational overhead. The related design is detailed in Chapter 6.

The approaches for decreasing the average response time in either single-leader (Fail-Stop) or BFT-enabled distributed control plane are presented in Chapter 5 and Chapter 7, respectively.

2.2.3 Configuration and Bootstrapping Complexity

Data-center SDN deployments heavily rely on OOBC, where control traffic is exchanged between switches and SDN controller(s) using dedicated links and switch management ports. OOBC, however, is often undesirable in large-scale and critical industrial environments due to its associated capital expenditure costs, complexity and installation efforts [38, 82, 176]. For example, deploying an additional network in machine tool manufacturing or drive technology networks is immensely expensive, due to associated cabling costs for the shielding required for Radio Frequency (RF) / Electromagnetic Interference (EMI) noise suppression. Similarly, avionics and automotive networks benefit greatly from decreased cabling weight [82]. Alternatively, the applicability of wireless in-band management is limited due to contention and medium unreliability [144, 163].

Industrial Ethernet protocols assume in-band control and management [224]. With In-Band Network Control (IBC), control traffic is forwarded along the same links used for the data plane traffic. This makes IBC networks the preferable solution from the cost and operational perspective. However, their implementation is challenging. For example, current popular open-source controller platforms ODL [123] and ONOS [39] provide no mechanisms to support or automate the IBC bootstrapping, nor do they allow for protection of control plane traffic against arbitrary link and node failures, both which are of critical importance in industrial networks [204]. Summarized, enablement of a distributed SDN control and its reliable bootstrapping in an industrial scenario must address the following challenges:

1. Establishing robust Switch-to-Controller (S2C) and C2C connections using IBC and an initially unconfigured control plane;
2. Preserving the control plane stability in case of multiple link, switch and / or controller failures;
3. Support for addition of new nodes and links to the previously reliably bootstrapped IBC network;
4. Minimization of manual per-device preconfiguration.

Our contribution: Chapter 8 investigates the design for automated bootstrapping of a resilient distributed control plane. We propose two novel automated bootstrapping schemes targeting IBC networks with highly-available control plane support and evaluate the proposed designs in realistic industrial topologies with varying sizes and controller placements. The deployments are evaluated w.r.t. *(i)* the

time required to converge the bootstrapping procedure, *(ii)* the time required to dynamically extend the network, and *(iii)* the flow table occupancy. The chapter also discusses general implementation aspects relevant for a successful introduction of the designs in industrial networks equipped with off-the-shelf OF agents.

Chapter 3

Enabling Fault Tolerance in Fail-Stop Control Plane

The centralized control paradigm of SDN introduces an SPOF and scalability challenges related to state synchronization of the control plane. A number of approaches have been proposed in literature to alleviate the SPOF issue [39, 100, 104, 183], with the majority relying on state- and function-replication across the replicas of the SDN controller cluster. With the concept of state replication, the SDN controller instances replicate their data store contents to other members that take part in a logical controller cluster, using a state distribution protocol of choice (e.g., Raft [87], Paxos [218]). When a failure of a controller is suspected, another replica from its cluster is able to take over and continue to serve future application's requests. In addition to fault-tolerance guarantees, deployment of the distributed control plane in industrial scenarios introduces requirements on guaranteed response time (ref. Section 2.2).

Chapter Summary: In this chapter, we discuss the distributed control plane design to tolerate Fail-Stop failures, deployed in existing controller designs and implementations of major open-source controller platforms ODL [123] and ONOS [39]. We henceforth introduce and present a SAN-based modeling framework for analytical estimation of resulting control plane availability when deploying state-of-the-art distributed SDN control plane solutions. We postpone the closely related discussion of the deterministic response time bounds during transient failures to Chapter 4, and in this chapter instead put focus on the availability-related evaluation only. To this end, we evaluate the availability of the distributed SDN control plane using long-term failure rates and steady-state analysis at sub-module, process and hardware level.

In the second part of this chapter, we demonstrate a correctness issue which may arise when leveraging distributed consensus. False positives in failure detectors of the controller replicas may lead to *oscillating leadership* and control plane unavailability and thus degrade the applicability of existing SDN controller designs in industrial environments. We first elaborate the problematic scenario. We then resolve the related issues by decoupling the failure detector from the underlying signaling methodology and by introducing *event agreement* as a necessary component of the proposed distributed control plane design. The proposed reference model is validated using exemplary implementations and extensions of the state-of-the-art mechanisms.

The proposed analysis methods and designs were published in augmented form in [8, 3].

3.1 Cold- and Hot-Standby Redundancy in SDN Control Plane

We coarsely distinguish two types of replicated controller deployments: (i) cold- and (ii) hot-standby redundancy.

Cold-standby redundancy: With cold-standby redundancy, a single controller instance is involved in control and management of all switches, operating autonomously until its eventual failure. Thus, the current *master* omits any communication with controllers from the set of *backup controllers* [128]. In the case of a master controller failure, a backup controller instance takes over the management of the switches in the impacted administrative domain.

This approach can be simply adopted in the industrial control plane [224] but comes with multiple limitations: (i) an operational interrupt is imposed due to a non-seamless take-over of the switches (i.e., registration / authentication of the new master controller in each switch); (ii) failure of the master results in the loss of any stateful controller information. Indeed, the backup controllers could observe the current state of the managed switches and thus fetch and synchronize their internal state in order to ensure consistency of control information in controllers' and switches' configuration data store. However, this comes with the penalty of a blocking synchronization period, as well as the loss of information on high-level network relations [94].

Consequently, practical state-of-the-art distributed SDN control plane solutions rely on *hot-standy redundancy* with active state synchronization during operation.

Hot-standby redundancy (Leader-based): With leader-based hot-standby redundancy, controller instances have their internal applications (i.e., routing, load-balancing) replicated. Similarly, the high-level configuration state (i.e., intent relations) as well as low-level configuration state (e.g., installed flow rules) are continuously replicated in each controller *replica* for the purpose of seamless and consistent fail-over. In order to provide a fallback solution in case of the current master controller's failure, the backup controllers keep track of the internal state information related to the switches managed by the master. When a master fails, another replica from the same cluster with an up-to-date state is elected the new master by the remaining members. Eventually, the elected new master takes over and resumes operation with some downtime.

Due to the majority of important industrial network control applications requiring stateful operation (i.e., resource and flow management for QoS-constrained operation [30, 9, 74, 15], inter-domain forwarding in TSN [224], intent translation and enforcement [39, 165] etc.), we focus on the hot-standby operation of controllers. We do note that in particular scenarios the requirement of consistent state exchange and synchronization among controller instances may indeed be relaxed to a state-independent (stateless) operation due to a decreased application complexity [1]. In the remainder of the thesis, however, we assume hot-standby replication as the measure of handling the stringest requirements of the industrial SDN control plane applications.

3.2 On Strong Consistency and Consensus

In the following subsections we give a definition of *strong consistency model*, enforced by the majority of existing distributed SDN control plane implementations, considered *necessary* by a subset of industrial applications. We furthermore elaborate on its relation to consensus algorithms. We then present an overview of the Raft consensus algorithm responsible for state synchronization, cluster leader election and state recovery of SDN controller replicas after failures. We put focus on Raft [87] due to its proven applicability in a number of practical SDN deployments [221, 222] and wide adoption in numerous open-source SDN platforms [39, 123].

3.2.1 Strong Consistency Model

With hot-standby replication model, controllers continuously synchronize their state in order to ensure consistent up-to-date state in each replica. Depending on the consistency model (i.e., strong [39, 78], eventual [35, 36, 113] or adaptive [32, 33, 2, 10]) which defines the ordering of synchronization messages, the synchronization procedure imposes a varying overhead on the control channel [10, 135]. The two major controller platforms ODL [123] and ONOS [39] implement the Strong Consistency (SC) model, which requires that the update of a distributed state has been seen by the majority of the cluster members prior to being committed. In the SC model, each consecutive operation that modifies the internal state of the controller is serialized and confirmed by a quorum of replicas, before processing subsequent transactions. Specifically, in leader-based SC approaches (e.g., in Raft [87, 146]), all requests are serialized by a cluster leader, in order to provide for a consistent data store view across all cluster followers. Thus, with SC, a large distributed system consisting of multiple controller replicas is effectively constrained into a monolithic system where each data store modification incurs two message rounds and a linear message complexity (in the case of a stable leader) in order to synchronize the controller views [48, 111, 133].

3.2.2 Consensus on Message Updates

In an SC cluster, whenever an update request is initialized by a client at one of the replicas, the receiving replica proxies the received request to the current cluster leader. The leader is the controller replica that orders all incoming state update requests, so as to allow for a serialized history of updates and thus operational state consistency at runtime. Following a state update at the leader, the update is propagated using a consensus protocol to the cluster replicas, and is committed to the data store only after the majority of replicas have agreed on the update. A consensus algorithm ensures that all replicas always decide on the same value (*agreement*), with the constraint that only a value proposed by one of the replicas eventually becomes accepted after the synchronization procedure (*integrity*). Google's Chubby [48] is a distributed locking service whose state-distribution and failure tolerance mechanism are based on a variation of the well-known Paxos consensus algorithm [218, 111]. ODL and ONOS, however, implement the more recent algorithm Raft [87]. Unlike Paxos, Raft also provides for persistent logging and state reconciliation for recovered replicas.

3.2.3 Assumptions on the Industrial Network

We assume a set C comprising $|C|$ SDN controller replicas, collected in a single cluster and deployed for the purpose of fault-tolerant operation. Fig. 3.1 depicts a deployment of the redundant control- and data planes in an exemplary industrial SDN comprising $|C| = 3$ controllers, and a number of disjoint paths in between them for fail-over purposes in case of link and node outages. In general, a deployment of $|C| = 2F + 1$ controllers tolerates a maximum of F controller failures before the SDN cluster becomes unavailable. Thus, in Fig. 3.1 only a single controller failure is tolerated before the cluster stops to serve the clients' requests. The clients of the controllers, such as the network administrators, switches, network appliances and end-hosts, can trigger controller events that lead to a cluster-wide state synchronization and subsequent event processing in the cluster *leader*. Clients communicate their requests (i.e., Remote Procedure Calls (RPCs), state updates, topology events etc.) to any live replica that is a member of the SDN cluster. The replica then contacts the current leader to serialize (order) the request, which in return distributes the request to other replicas, commits the request and executes its local state machine (in zero-failure case). The replica is then notified of the result of the request execution and can respond to the client with an application response. In the case of a leader failure during the request processing, a new leader is elected by executing a consensus algorithm and the synchronization process is re-initiated.

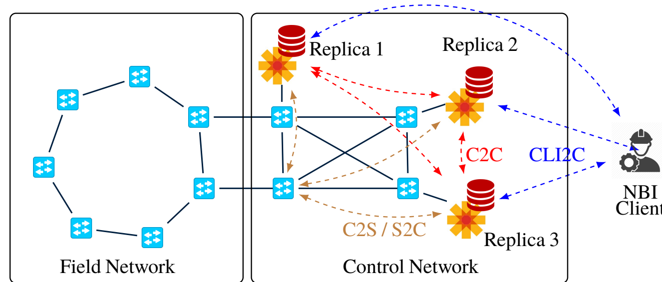


Figure 3.1: Exemplary industrial SDN with redundant paths for majority of Controller-To-Controller (C2C) and Controller-To-Switch (C2S) connections. SDN controllers execute a consensus protocol (e.g., Raft [87]), responsible for state synchronization, leader election and cluster recovery after individual replica failures. Red dashed lines represent the consensus protocol exchanges between the replicas, blue dashed lines are the "client" connections (S2C or northbound Client-to-Controller (CLI2C)).

The limitation of supporting only F failures when $2F + 1$ replicas are deployed relates to the Consistency, Availability, Partition Tolerance (CAP) theorem [68]. This theorem states that any distributed system can provide a maximum of two of the following three system properties at the same time: consistency, availability and partition-tolerance. Consensus algorithms such as Raft [87] and Paxos [218, 111] favor consistency and partition-tolerance properties, and hence can forward their state consistently even in the face of network partitions. A consistent operation of a controller cluster ensures that the majority of controllers have the same controller state at any given time, and that no two conflicting state updates are ever successfully committed to the shared update history. Hence, controllers are in consensus with regards to their state. Consistent and partition-tolerant operation, however, comes at the cost of a decreased service availability, since consistent operation in the face of network partitions can only be guaranteed by disabling the operation of minority partitions while

the majority continues to operate. Thus, we consider the system *available* whenever the majority of replicas are available and are mutually reachable.

Controller-to-Switch Channel: Typically, the SDN controllers implementing hot-standby synchronization also implement the OF-based [122] Southbound Interface (SBI) to control and manage OF-based data plane elements. OF allows for assignment of multiple controller instances to the same switch, using IP-based controller list configuration. Thus, multiple controllers may concurrently connect to the same switch. However, depending on the selected operation mode, either all or a maximum of one assigned controller (i.e., the switch's master) may modify switch configurations at any point in time. To support consistency of the control plane, production-ready distributed controller platforms typically allow only the current master controller to modify the switch state at any point in time.

3.2.4 Related Work - On Availability Estimation of a Distributed SDN Control Plane

Availability and overhead modeling of SDN has recently started to gain traction. In [141, 142], Nencioni et al. investigate the impact of operational and management failures on the availability in SDN. They focus on the long-term availability impact of adding additional controllers, but do not consider the impact of C2C synchronization at micro-scale nor do they provide for a response time analysis (ref. Section 3.3).

Tuncer et al. [184] propose a placement heuristic to cater for the optimality of the controller cluster imbalance. Given an arbitrary network topology, they compute the number of required replicas and their placement, while considering a *distance* constraint (i.e., the C2C delay). We instead focus on the complementary problem of the availability analysis of an arbitrary SDN cluster configuration / placement.

Several works have investigated the phenomenon of *software aging* wherein the health of a software system degrades over time [185, 194]. These works conclude that a mechanism which *rejuvenates* the software component to its stable state, would provide long-term benefits in terms of its availability. Accordingly, in Section 3.3.3 we introduce the mechanism for fast software system recovery that relates to the concept of *software rejuvenation*. We have evaluated the benefits of the *reactive* controller recovery where, following a detected failure, the impacted component is reinitialized in order to minimize the downtime. While considered in research, the alternative *proactive* software rejuvenation is currently not featured in any of the existing SDN controller implementations. An implementation of the respective proactive mechanism would require sophisticated aging-detection characteristics [185], which in return requires considerable effort to become practical in complex controller codebases that potentially span millions lines of code [230].

Since all available SDN cluster implementations focus on a single master for any switch in its administrative domain at runtime, we put focus on the evaluation of a single-leader Raft-based cluster and consider its direct comparison with multi-leader approaches such as EPaxos [133] as future work. Comparison of Raft with *eventually consistent* approaches [10, 113] in terms of the resulting control plane response time is presented in Chapter 5.

3.2.5 Raft Working Model

Raft is a distributed consensus algorithm that provides safe and ordered updates in a system comprised of multiple running replicas. Raft is the only consensus algorithm implementation in ODL and ONOS. It solves the issues of understandability of the previous de-facto standard consensus algorithm Multi-Paxos [111], and additionally standardizes the leader election and post-failure replica recovery.

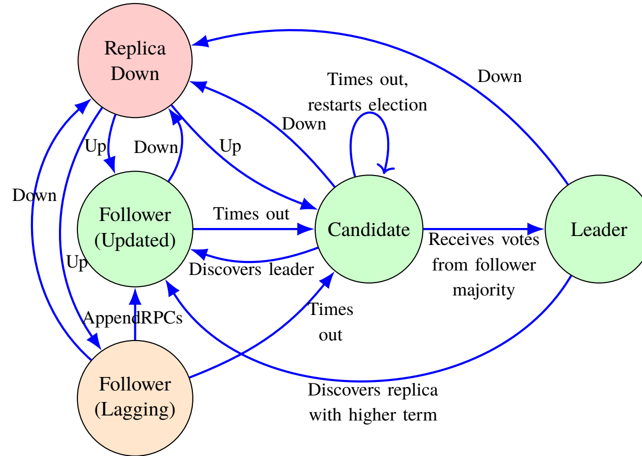


Figure 3.2: A simplified lifecycle schema of a replica inside the Raft cluster. Adapted from [87], extended for the purpose of detailed modeling.

A Raft cluster comprises *leader*, *follower* and *candidate* replica roles. The leader is the node that parses and distributes incoming client updates (*reads*, *writes* and *no-ops*) to Raft followers and ensures safe commits. The majority of cluster followers must confirm the acceptance of a new update before the leader and the followers may commit the update in the local *commit log*. Only after the update is committed, the SDN applications built on top of a Raft agent can continue their processing. After the application has executed the operation associated with the state update, a response is forwarded to the requesting client (e.g., a switch or an NMS). Raft guarantees that the applied state updates are eventually committed in every available replica in the cluster in the right order. Furthermore, each update is applied *exactly once*, hence enabling *linearizable semantics* [87] when operating on the controller state. In the case of a leader failure, after an expiration of an internal follower timeout, the remaining followers automatically switch to a candidate role. A *candidate* is an active replica which offers to become the new cluster leader. To do so, it propagates its candidate status to the other available replicas. If a majority of nodes vote for the same candidate, this candidate node becomes the new leader.

Updates in Raft require a single-round trip delay between the leader and the *preferred follower majority* (the fastest to reach followers). When a controller failure occurs, depending on the role of the failed replica, additional delay overhead is imposed. Failures in the Raft leader during the processing of a particular update lead to a new leader election after an expired *election timeout*. After an exceeded *client timeout*, the client retries its request. If instead of the leader a follower had failed, depending on the follower's type *and* the number of active followers, we distinguish three scenarios:

- Failure of a follower that *is not a member* of the preferred follower majority results in no additional imposed delays between the leader and the cluster majority.
- Failure of a follower that *is a member* of the preferred follower majority leads to the Raft leader having to include an additional "slower" follower in the preferred follower majority. This, in return, negatively impacts future update commit times. The intensity of added delay depends on the follower's placement and distance to the leader.
- Failure of any follower that comprises the follower set, with no backup followers available (stand-by Raft members), necessarily leads to the cluster unavailability and thus rejection of incoming client requests.

After a successful recovery of the majority of the Raft members and the re-election of a new leader, Raft is able to forward its state and commit new updates to the commit log. Depending on the source of failure and the repair time, as well as on the Raft recovery parameters (*candidate* and *election timeout*), the recovery takes a non-deterministic period to finish.

Fig. 3.2 gives a high-level overview of the states a cluster replica may traverse throughout its lifecycle. We present the more detailed structural and behavioral models of Raft in zero- and multiple-failure cases in Section 3.3.2.

3.3 Modeling the Availability of a Raft-enabled SDN Control Plane

In this section, we first give a brief summary of the most important SANs concepts used in our modeling of a distributed SDN control plane. We then proceed to provide detailed estimation of availability measures of an industrial control plane. Our SANs comprise the detailed sub-models of the Raft consensus algorithm, specifically the cluster failure and recovery processes of Raft. The response handling model, on the other hand, is detailed in later Section 4.1. Failures are modeled as stochastic arrival processes for long-term, and deterministic occurrences for worst-case evaluations. As per the nature of the modeled consensus algorithm Raft, the recovery process too is a combination of stochastic and deterministic message and timeout delays. We further introduce an enhancement to the current controller platforms for enabling a fast recovery of controller bundles and processes. We evaluate its benefits w.r.t. to the long-term cluster availability.

Our SAN models are compiled into Continous Time Markov Chain (CTMC) state spaces. In contrast to existing works on consensus algorithms that derive their performance analysis from experiments, we provide analytical guarantees. To this end, our numerical solutions cover the space of all possible state combinations which an SDN cluster may be in.

3.3.1 Overview of Stochastic Activity Networks (SANs)

In contrast to measurement techniques, *deductive* analysis allows for a system evaluation *before* the system is actually deployed. Hence, significant savings can be achieved if the deductive solutions are able to accurately predict the real-world behavior of the future non-implemented system or system

extensions. *Discrete-event simulation* is, for example, partially applicable to our problem. Simulation allows for a tunable quality of the results by repeating execution of a given model and derivation of the relevant output measures. However, the simulation methodology may not handle corner cases, which are numerous in a consensus algorithm such as Raft. With this in mind, contrary to the previously published methods on evaluation of the distributed SDN control plane, which base their analysis on a limited number of physical cluster configurations [181, 182], we opt for a deductive study.

Analytic numerical solvers allow for an accurate evaluation of each system state configuration. For this purpose, they require a manually or automatically generated model state space as an input. The additional overhead of the state space generation, as well as the inclusion of each state in the solution, generally leads to a higher computational effort compared to simulation. Namely, the generation of the state space may lead to a state explosion problem and infeasible solving times. Therefore, in Section 4.1.4.1 we discuss the scalability of the presented models. Instead of manual state modeling, we automate the model generation process and hence avoid the issue of *largeness* [44] of the resulting state space. For the purpose of automated model generation, we rely on SANs, a prominent representative of model generation frameworks.

SANs are an extension of Petri Nets (PNs) and an established graphical language for describing the system behavior. They have been successfully used in survivability and performability studies of critical infrastructures [34], industrial control systems [140] and telecommunication systems [71] since the late 1980s. We specifically choose SANs over Generalized Stochastic Petri Nets and Stochastic Reward Nets due to their practical extensions for the *inhibition* of state transitions, as well as the flexible predicate assignment to the *gate* abstractions.

A SAN consists of *places*, *activities*, *input gates* and *output gates*. Places have a certain *token* assignment associated with them. Every unique assignment of tokens across the places uniquely defines a state of the SAN. These states are called *markings*. In Markov Chain analogy, a single marking represents a unique state of a Markov Chain. An *activity* element of a SAN defines a transition with the corresponding transition rates, and allows for controlling the flow of tokens from a single SAN place into a different SAN place. Furthermore, an activity allows for connecting a place to an *output gate* where, on transition of a token from a place to an output gate, a sequence of actions can be taken - e.g., "*if the number of tokens in place A >n → increment the number of tokens in place B by m*". Hence, compact state changes (and a large number of unique markings) triggered by a particular transition may be modeled using a smaller number of modeling elements, compared to a traditional Markov Chain.

When an activity *fires*, a number of tokens are removed from the source place and transferred to a destination place connected by the activity. An *input gate* serves as an inhibitor of an associated activity. It specifies a boolean predicate which, when evaluated *true*, enables an activity and allows the firing of the activity. If the inhibitor evaluates *false*, the associated activity is disabled. An *instantaneous* activity is enabled at all times, and will fire whenever there are tokens available in its input place. A *timed* activity, on the other hand, is assigned a *time distribution function* which specifies the firing rate of a specific activity. In our model, for timed activities we assign the deterministic (Erlang-approximated) and exponential firing rates, but also specify instantaneous activities where

necessary. An activity may further lead to a token transfer from a source place to one of multiple destination places. This uncertainty is modeled using a *case* definition for each destination state, where each case is assigned a probability parameter.

To solve the SAN, it is first transformed into a discrete-state stochastic process [44]. We make use of the flat state space generator implemented in the Möbius modeling tool [55], to generate the CTMC state space inherent to the evaluated SAN. To derive instantaneous state probabilities of a CTMC, the transient solver of Möbius implements the *uniformization* method [44, 156]. In short, using uniformization, the *transient state probability vector* $\pi(t)$ of the CTMC can be expressed in terms of a one-step probability matrix of a Discrete Time Markov Chain (DTMC), so that all state transitions of a resulting DTMC occur with a *uniform* rate q. As a result of the transformation, the desired state probability vector $\pi(t)$ at time t is governed by a Poisson variable $q \cdot t$ and can be expressed as follows:

$$\pi(t) \approx \sum_{i=l}^{i=r} v(i) \cdot e^{-q \cdot t} \frac{(q \cdot t)^i}{i!} \text{ where } v(0) = \pi(0) \tag{3.1}$$

where $v(i)$ represents an iteratively computed DTMC state probability vector at step i. Lower and upper bounds, l and r, govern the number of iterations required to compute the state probability vector with an overall error tolerance of $\varepsilon = \varepsilon_l + \varepsilon_r$ and truncation points l and r, respectively.

We refer the reader to comprehensive descriptions of SANs in [44, 162].

3.3.2 SAN Model Representations

In this section, we present the SAN models for failure and recovery processes of a Raft-enabled SDN control plane. We represent *places* as blue circles, *timed activities* as thick blue vertical bars, *instantaneous activities* as thin blue vertical bars, and *input* and *output gates* as thick red and black arrows, respectively.

3.3.2.1 Cluster Failure Model

To evaluate the performance of the SDN distributed control plane and Raft in the face of failures, we introduce a dedicated failure model. For general long-term considerations, we distinguish between hardware and software failures with failure rates λ_{F_H} and λ_{F_S}, respectively. All specified non-deterministic timeouts, failure and repair rates in our model follow a Negative Exponential Distribution (N.E.D.). For software failures, we distinguish failures at the application bundle (i.e., an Open Services Gateway initiative (OSGi) bundle in ONOS [39] and ODL [123]) and process level. Similarly, repair rates are distinguished as specified in Table 3.1.

The SAN failure model is depicted in Fig. 3.3. The place `NodesUp` contains the total number of available controllers, not yet necessarily assigned a Raft member role. Depending on the failure type (at hardware, process or bundle level), after an occurrence of a failure, a token is placed into the respective `NodesDown` place. Furthermore, each firing of a failure activity triggers a token addition in the place `NodeDownSelectFailure` and results in a subsequent evaluation in the instantaneous *case* activity `failureSelectRole`. We distinguish between the safe-follower (F_{Sf}), follower-majority

(F_{Mj}) and leader failures (F_{Ldr}), with probabilities:

$$P(F_{Sf}) = \begin{cases} \frac{F_{up}}{F_{up}+1} & \text{when } F_{up} \geq \left\lceil \frac{|C|-1}{2} + 1 \right\rceil \wedge L_{up} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$P(F_{Mj}) = \begin{cases} 1 & \text{when } F_{up} < \left\lceil \frac{|C|-1}{2} + 1 \right\rceil \vee L_{up} == 0 \\ 0 & \text{otherwise} \end{cases}$$

$$P(F_{Ldr}) = \begin{cases} \frac{1}{F_{up}+1} & \text{when } F_{up} \geq \left\lceil \frac{|C|-1}{2} + 1 \right\rceil \wedge L_{up} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where F_{up} and L_{up} are the counters of tokens in places `FollowersUp` and `LeaderUp` before the failure occurrence, respectively. Failure of the follower-majority F_{Mj}, or a leader failure F_{Ldr} during the event handling in controller, results in a client timeout and subsequent restart of the event handling process. On the other hand, failure F_{Sf} does not affect the cluster availability as the reorganization of a stable cluster majority is still possible with the remaining nodes, albeit with an added delay (as per the definition of T_M, discussed in the SAN-based response time analysis in the next chapter).

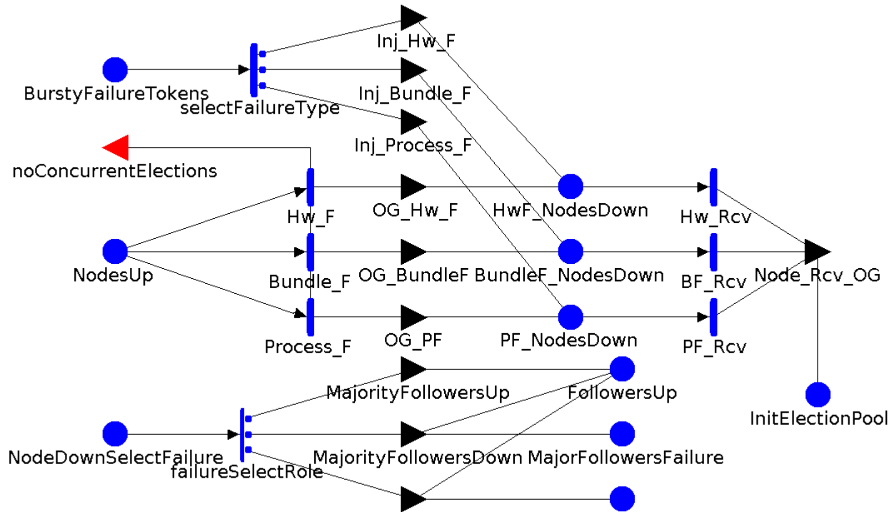


Figure 3.3: SAN model of the failure processes includes the long-term failure rates (`Hw_F`, `Process_F` and `Bundle_F`) and the controlled failure injection (`Inj_Hw_F`, `Inj_Process_F` and `Inj_Bundle_F`). The failure type is decided based on a random selection process (bottom-left), and its severity is a function of the current system state (bottom-right).

In order to observe the response time during and shortly after the failure, we also model a procedure for *controlled failure injection* of single and multiple-correlated transient controller failures and observe the system performance over a short-term time range at millisecond granularity. The correlated failures are modeled as bursty and may occur concurrently. In the past, correlated failures have been investigated in the context of distributed systems [137], and represent a flexible method to consider chained failure propagation, i.e., resulting from a malfunctioning replicated SDN application. The failure injection process is depicted in the upper left part of the SAN shown in Fig. 3.3. The place `BurstyFailureTokens` initially holds a number of tokens corresponding to the number of

simultaneous bursty failure injections. The activity `selectFailureType` governs the probability distributions for encountering a particular type of node failure.

Critical data plane failures: In a DTMC, the probability of occurrence of an SDN controller element failure F_c or a critical data plane element failure F_d corresponds to:

$$P(F_c \cap F_d) = P(F_c) + P(F_d) - P(F_c \cup F_d) \tag{3.2}$$

In the continuous time domain, failure arrivals for the critical data plane elements that carry the C2C flows, and failure arrivals for the SDN controller elements can be represented as two independent Poisson processes $N_d(t)$ and $N_c(t)$ with the unique firing rates λ_d and λ_c, respectively. Since the two processes are independent, they also have independent increments. Therefore, critical failure arrivals associated with the summed process $N_t(t) = N_d(t) + N_c(t)$ can be modeled using the rate $\lambda_t = \lambda_d + \lambda_c$.

The failure rates for the critical data plane paths which carry the network control flows can be embedded in the parametrization of our models without an additional modeling overhead (ref. Table 3.1). However, we primarily focus on studying the control plane consensus for the use case of a highly redundant industrial network [130, 131]. Thus, we intentionally decouple our work from the data plane reliability studies and assume the reliable parametrization $1/\lambda_d = \infty$.

3.3.2.2 Cluster Recovery Model

The Raft recovery SAN model in Fig. 3.4 depicts the process of re-inclusion of a previously disabled controller replica in the Raft cluster. The place `InitElectionPool` holds a token for each running controller replica that is available but still needs to be admitted in the cluster. As per Raft design, the replica expects the Raft leader of the current term to announce its presence using a leader heartbeat. If a leader is identified before the *follower timeout* expires, the replica takes upon the follower role and a token is assigned to the place `AnnounceFollowerRole`. Alternatively, the replica switches to the *candidate* role (place `AnnounceCandidateRole`). Three cases are now possible, each adding its specific delay to the overall response time:

1. If the cluster majority is up ($\geq \lfloor \frac{|C|}{2} \rfloor + 1$) and the replicas acknowledge the candidate as a new leader before the expiration of the *candidate timeout*, the candidate is elected as the leader (output gate `setLeaderUp`). The announcement of the candidate role from the candidate to the cluster majority takes an additional round trip.

2. If another leader is identified while the replica is in the candidate state, the candidate replica becomes a follower and a token moves from the `AnnounceCandidateRole` place to the `setNewFollowerUp` output gate.

3. If the cluster majority sends no acknowledgment to the candidate nodes during the candidate timeout (occurs whenever a total of $\left\lfloor \frac{|C|}{2} \right\rfloor + 1$ replicas are down), the candidate waits for the timeout to expire and then repeats the candidate procedure.

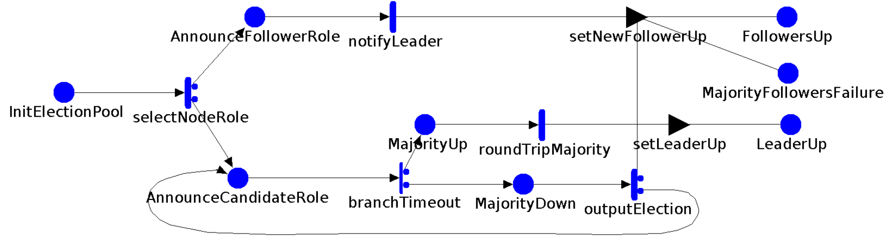


Figure 3.4: The Raft recovery SAN depicts the inclusion of a previously unavailable controller replica into the cluster. The replica may become either the new Raft leader or a follower. Duration of the recovery process will affect the resulting event response time and the cluster availability if the recovering replica is needed to establish a follower-majority and elect a new Raft leader.

If the replica becomes a Raft leader, a token is assigned to the `LeaderUp` place (previously empty), alternatively the token is assigned to the place `FollowersUp`. In both cases, the `NodesUp` token counter is incremented by 1.

3.3.3 Fast Recovery Mechanism for Bundles and Processes

In contrast to evaluating the system with purely fixed software repair rates [117, 141], we model the recovery process taking into account state-of-the-art SDN controller implementations. Furthermore, we propose an optimization to enhance upon the standard repair time - a *watchdog*-like mechanism that monitors the critical controller components' health and correctness. The watchdog monitors both the granular SDN controller bundles and the controller process (comprising many bundles). Whenever a bundle or a process fails, watchdog schedules a rejuvenation procedure [185] that repairs the affected component.

Realization: While there may exist various designs to realize a watchdog functionality for the purpose of monitoring the liveness of a software or hardware component, we opted to implement the watchdog as a software agent external to the OSGi container hosting the SDN controller bundles. Following a successful start-up of both the watchdog and the SDN controller processes, the watchdog establishes a connection to controller's OSGi environment. We make use of the Apache Karaf's[1] *Remoting* mechanism to allow for remote connections to a running Karaf instance.

Our agent periodically polls the status of a bundle's lifecycle and discovers that the bundle is in one of the following UP states: {*INSTALLED, STARTING, ACTIVE*}; or DOWN states: {*UNINSTALLED, STOPPING, RESOLVED*}. Upon discovery of a bundle that is in a DOWN-state, the agent schedules a *bundle:start*-transition for the affected bundle, in order to get it up and running in an UP-state. In the case of an unsuccessful remote connection to Karaf, the watchdog evaluates the current list of processes for false positives and, if a missing Karaf process is detected, it schedules an immediate restart of Karaf.

The watchdog process could also be executed externally to the machine running the SDN controller. Hence, while not considered in our evaluation, the same mechanism can be applied to schedule physical

[1]Apache Karaf - an OSGi distribution offered by the Apache Software Foundation based on Apache Felix - https://karaf.apache.org

or Virtual Machine (VM) reboots in case of a hardware or hypervisor failure. On the other hand, hardware or hypervisor issues may be a sign of misconfigurations or recurring defects whose source should be diagnosed manually.

To collect the accurate real-world repair rates for controller bundles and processes, we have used our watchdog agent implementation to evaluate the bundle and process reboot times in a clustered ODL setup. We had experimentally injected bundle- and process-critical failures in sequence and then measured the subsequent recovery time required to re-stabilize the system. The distinguished mean bundle and software process repair times, measured during the controlled rejuvenation of the critical Raft component *sal-distributed-datastore* and the ODL's controller process, peaked at $182.9ms$ and $26.9s$ respectively, far below the 3 minute recovery intervals previously proposed in literature [141, 186]. The measured recovery time purposely does not include the time needed to re-include the recovered node in the Raft cluster, since this is modeled as a separate non-deterministic process in our SANs. The bundle and process reboots took place inside a dedicated ODL VM that was part of a bigger ODL controller cluster, virtualized on a modern Intel Xeon-based server, with each of the ODL VMs assigned 4 vCores and 8 GB of DDR4 memory. ODL was loading the OSGi bundles available in the *OpenFlowPlugin* and *Controller* projects and had the *Clustering* component enabled.

Table 3.1: SAN model parameters used in our solutions.

Parameter	Intensity	Unit	Meaning
T_A	1	[ms]	Application handling time
T_C	1	[ms]	Data store commit delay
$1/\lambda_f$	225	[ms]	Mean follower timeout
$1/\lambda_{ca}$	225	[ms]	Mean candidate timeout
T_{CL}	50	[ms]	Client timeout
$T_R(T_{M_{worst}})$	10	[ms]	Worst-case replica-leader delay
$T_{M_{best}}$	5	[ms]	Best-case majority-leader delay
T_{CR}	1	[ms]	Delay client-to-replica
N_F	$1..C$	N/A	Controller failure count
$1/\lambda_{F_H}$	6	[months]	Hardware failure rate
$1/\lambda_{F_S}$	1	[week]	Software failure rate
$1/\lambda_{F_{S_i}}$	$30/\#_F$	[ms]	Software failure rate (injected)
$1/\lambda_{R_H}$	12	[h]	Hardware repair rate
$1/\lambda_d$	∞	[h]	Critical data plane failure rate
$1/\lambda_{R_S}$	3	[minutes]	Bundle and process repair rate
$1/\lambda_{R_{Sbw}}$	182.9	[ms]	(*Watchdog*) Bundle repair rate
$1/\lambda_{R_{Spw}}$	26.9	[s]	(*Watchdog*) Process repair rate
E_s	20	N/A	Erlang approximation stages
R_M	10	N/A	Max. inconsistent Raft terms

3.3.4 Evaluation and Results: Raft Cluster Availability

Fig. 3.5 depicts the *unavailability* of a 3-node controller cluster setup. We emphasize the long-term advantage of an SDN controller bundle / process watchdog mechanism by evaluating the availability of 3-node configuration over an observation period of 1000 hours. The unavailability measure is defined as the probability of encountering an unavailable cluster of controllers at any time instant t as: $P_{C_U}(t) = 1 - P_{C_A}(t)$. $P_{C_A}(t)$ represents the probability of encountering a system in a state where the Raft leader and the majority of Raft followers are available and have converged their leader-election

processes. Software and hardware failures are modeled using the long-term exponential hardware and software failure rates presented in Table 3.1.

The approximated unavailability measure saturates after ~85 hours, which is an expected mean failure time for the combined software failures at bundle and process level, given the individual exponentially distributed failures with a mean of 1 week (~170 hours) for individual arrivals. We consider the process and bundle failure arrivals as two independent Poisson processes with variably configured rates. Hence, merging the two independent processes with equal arrival rates results in an approximately halved inter-arrival time between software failures. The usage of a watchdog that proactively rejuvenates a system after a software failure leads to a shorter overall experienced downtime, and hence a lower expected Raft cluster unavailability in the long-term. Configurations with five or more replicas guarantee a negligible unavailability of $< 1e^{-9}$ and are hence not included in the figure.

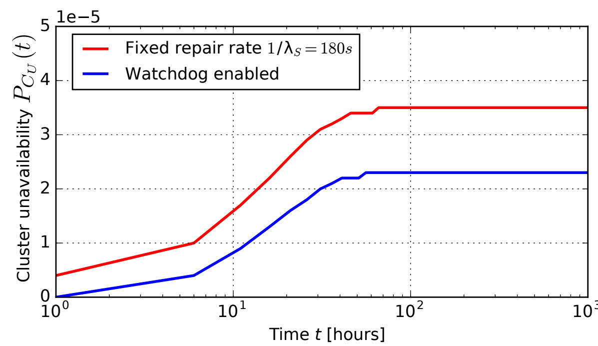


Figure 3.5: Transient analysis of the SDN controller cluster unavailability over a 1000 hour period. The cluster size of 3 controllers was considered in the analysis. As expected, the inclusion of a liveness guard mechanism results in a lower overall unavailability. SDN controller clusters that include 5 or a higher number of replicas per cluster have shown to posses negligible availability concerns. This confirms the claims made in [141], where authors discuss the minimal effect of long-term failure rates on the experienced downtime of the control plane.

3.4 Supporting Partially-Observed Network Partitions: Decoupling Consensus from Event Detection

In Raft, the leader is in charge of serialization of message updates and their dissemination to the followers. In the face of network partitions and unreliable communication channels between particular replicas, however, the simple timeout-based failure detector of Raft cannot guarantee the property of the stable leader [201]. Zhang et al. [201] demonstrate the issue of the oscillating leadership and unmet consensus in two scenarios involving Raft. To circumvent the issues, the authors propose a solution - redundant C2C connections with packet duplication over disjoint paths. This solution is expensive for a plethora of reasons: a) additional communication links impose significant communication overhead and thus negatively impact the system scalability; b) selection of the paths that would alleviate the failures is not a trivial task, especially when the locality and the expected number of link / node failures is unknown; c) certain restrictions on topology design are imposed - their solution requires disjoint paths between any two controller replicas.

In contrast to [201], we identify and associate the core issue of unreliable Raft consensus with the lacking design of its failure detection. In fact, state-of-the-art control plane solutions, i.e., ODL [123], Kubernetes [143], Docker (Swarm) [139]) all tightly couple the Raft leader election procedure with the underlying follower-leader *failure detection*, thus allowing for false failure suspicions and arbitrary leader elections as a side-effect of network anomalies.

We solve the issue of oscillating consensus leadership and unmet consensus as follows: We alleviate the possibility of reaching false positives for suspected controller replicas by disaggregating the process of distributed agreement on the failure event, from the failure detection. Namely, we require that an agreement is reached among active controller processes on the status of the observed member of the cluster, prior to committing and reacting to a status update. We confirm the correctness of the analytical formulation for the worst-case agreement period using an empirical approach.

3.4.1 Related Work - On Correctness Issues with Raft

Zhang et al. [201] propose packet duplication for C2C connections over disjoint paths to solve the issues of *oscillating leadership* and *unmet consensus* in scenarios involving Raft. In comparison, we solve these issues more efficiently by removing the possibility of reaching false positives for suspected controller replicas by introducing the requirement of distributed agreement on the failure event.

In another recent evaluation of Raft in ONOS [39], Hanmer et al. [78] confirm that a Raft cluster may continuously oscillate its leader and crash under overload. This behavior may arise due to increasing computation load and incurred delays in heartbeat transmissions in the built-in signaling mechanism. The issue is unsolvable using the means of disjoint flow replication alone [201].

On a similar note, in Raft, a *split votes* scenario may occur after a leader failure and a subsequent simultaneous leader election procedure. In *split votes*, the nodes in the cluster compete for the cluster leadership [87] without any of the nodes winning the majority vote. The candidates are hence unable to come to consensus about the future leader since the votes remain repeatedly divided. To minimize the probability of a *split vote*, Raft introduced randomized *follower timeout* parametrization which varies the per-node time-to-become-candidate duration ($\sim 150ms - 300ms$ as per [87]). The split votes may theoretically delay the leader election indefinitely, however, designs such as a Round-Robin-based election of the leader or randomized timeouts ensure the scenario does not occur in practice.

3.4.2 Problematic Scenario: Oscillating Leadership

With single-leader consensus, e.g., in Raft-based clusters, the controller replicas agree on a maximum of a single *leader* during the current round (called *term* in Raft). The leader replica is in charge of: i) collecting the client update requests from other Raft nodes (i.e., *followers*); ii) serializing the updates in the underlying replicated controller data store; iii) disseminating the committed updates to the followers.

In the case of a leader failure, the remaining active replicas block and wait on leader heartbeats for the duration of the expiration period (i.e., the *follower timeout* [87]). If the leader has not recovered, the follower replicas announce their candidation for the new leader for the future term. The replicas assign

their votes and, eventually, the majority of the replicas vote for the same candidate and commonly agree on the new leader. During the re-election period, the distributed control plane of the system is unavailable for serving incoming requests [3]. However, not just the leader failures, but also unreliable follower-leader connections as well as link failures, may lead to arbitrary connection drops between members of the cluster and thus the re-initialization of the leader election procedure [201]. Raft realizations typically implement the C2C synchronization channel as a point-to-point connection, thus realizing $\frac{|C|\cdot(|C|-1)}{2}$ bidirectional connections per replica cluster.

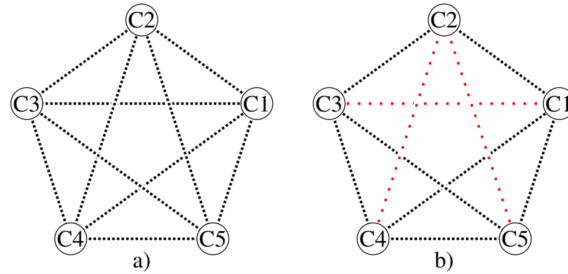


Figure 3.6: *Non-failure* (a) and an exemplary *failure case* (b) scenarios with injected communication link failures (loosely dotted). In the non-failure case, direct any-to-any connectivity between the controller replicas is available. The depiction is agnostic of the underlying physical topology and represents a logical connectivity graph.

To demonstrate a possible issue with the consensus design, consider the task of reaching consensus in the multi-controller scenario comprising $|C| = 5$ controllers, depicted in Fig. 3.6 a):

1. Assume all depicted controller replicas execute the Raft [201] consensus protocol. Let replica `C2` be selected the leader in term `T1` as the outcome of the leader election procedure.

2. Assume multiple concurrently or sequentially induced link failures, i.e., on the logical links `(C2,C4)` and `(C2,C5)`. Following the expiration of the *follower timeout*, `C4` and `C5` automatically increment the current term to `T2` and initiate the leader re-election procedure by advertising their candidate status to `C1` and `C3`. Since a higher-than-current term `T2` is advertised by the new candidate(s), `C1` and `C3` accept the term `T2`, eventually vote for the same candidate, resulting in its election as leader. Depending on whose follower timer had expired first, either `C4` or `C5` is thus elected as the leader for `T2`.

3. After learning about `T2`, `C2` steps away from its leadership and increments its term to `T2`. As it is unable to reach `C4` and `C5`, `C2`'s *follower timeout* eventually expires as well. Thus, `C2` proceeds to increment its term number to `T3` and eventually acquires the leadership by collecting the votes from `C1` and `C3`. Term `T3` begins. The **oscillation of leadership** between `C4/C5` and `C2` can thus repeat indefinitely.

4. Should either `C4/C5` or `C2` individually apply a client update to their data store during their leadership period, the replicas `C1` and `C3` will replicate these updates and thus forward their log correspondingly. This in return leads to the state where only replicas `C1` and `C3` are aware of the

most up-to-date data store log at all times. As per the rules of Raft algorithm [223], from then onwards, only C1 and C3 are eligible for leadership.

5. Since the link / connection (C1,C3) is also unavailable (ref. Fig. 3.6 b)), the leadership begins to oscillate between replicas C1 and C3, following the logic from Steps 1-3.

Hence, in the depicted scenario, the Raft cluster leadership oscillates either between C2 and C4 / C5, or C1 and C3. Zhang et al. [201] have empirically evaluated the impact of the oscillating leadership on system availability. During a 3-minute execution of an oscillating leadership scenario using LogCabin's[2] implementation of Raft and 5 Raft nodes, the authors observed an average *unavailability* of ~58% and 248 leadership shifts, even though the network itself was not partitioned.

3.4.3 Decoupling Consensus from Failure and Recovery Detection

We next introduce a generic design that alleviates the leader oscillation issue. Specifically, we ensure the above Step 2 never occurs, by requiring all active replicas to reach agreement on the inactive replica prior to restarting the leader election.

The communication topology between replicas is modeled by a connectivity graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ where $\mathcal{C}$ is the set of controller replicas and $\mathcal{E}$ is the set of active *communication links* between the replicas. An active link (i, j) denotes an available point-to-point connection between the replicas i and j. The communication link between two replicas may be realized using any available mean of data-plane forwarding, e.g., provisioned by OF flow rule configuration in each hop of the path between i and j, or by enabled Media Access Control (MAC)-learning in non-OF data plane.

Let $\mathcal{D}$ contain the guaranteed worst-case delays between any two directly reachable replicas. In the *non-failure case* (ref. Fig. 3.6 a)), $\forall(i, j) \in \mathcal{E} : \exists d_{i,j} \in \mathcal{D}$. In the *failure case*, depicted in Fig. 3.6 b), we assume that partitions in the connectivity graph have occurred (e.g., due to flow rule misconfigurations or link failures). Connectivity between the replicas i and j may then require message relaying across (multiple) other replicas on the path $\mathcal{P}_{i,j}$, with $\mathcal{P}_{i,j} \subseteq \mathcal{E}$. In both *non-failure* and *failure* cases, direct communication or communication over intermediate proxy (relay) replicas, respectively, is possible between any two active replicas at all times.

Replicas that are: a) physically partitioned as a result of one or more network failures, and are thus unreachable either directly or over a relayed connection by the replicas belonging to majority partition; or are b) disabled / faulty; are eventually considered *inactive* by all active replicas in the majority partition.

Each controller replica executes a process instance incorporating the components for *failure detection* and *event agreement*, together with the corresponding *signaling* and *dissemination* methods (Section 4.2.3 details these components in more detail). Any *correct* proposed combination of these methods must fulfill the properties of:

- *Strong Completeness*: Eventually all inactive replicas are suspected by all active replicas.

[2] LogCabin - Distributed storage system built on Raft - https://github.com/logcabin/logcabin

- *Eventual Strong Accuracy*: Eventually all active replicas (e.g., still suspected but already recovered) are not suspected by any other active replica.

3.4.4 Introducing the Event Agreement Component

The replica i can become suspected as inactive by replica j, following either the failure of the observed replica i or the failure of the communication link (i, j). Since individual replicas may lose connection to a particular non-failed but unreachable replica (i.e., as a consequence of an active replica but failed physical link(s) or data plane node(s) undermining the logical communication link (i, j)), a particular subset of replicas may falsely consider active replicas as inactive. To alleviate this issue, we introduce the Event Agreement function block in each replica.

To reach agreement on an observed replica's state, the active observing replicas must first acknowledge the failure of the suspected replica in an agreement phase. We consider two types of the event agreement - "*local*" and "*global*" agreement:

- *Local Agreement* on failure (recovery) is reached when a replica observes that at least $\lceil \frac{|C|+1}{2} \rceil$ *active* replicas have marked the suspected replica as `INACTIVE` (`ACTIVE`).
- *Global Agreement* is reached when a replica observes that that at least $\lceil \frac{|C|+1}{2} \rceil$ *active* replicas have confirmed their *Local Agreement* for the suspected replica.

The global agreement imitates the semaphore concept, so to ensure that active replicas have eventually reached the same conclusion regarding the status of the observed replica. We assume that physical network partitions may occur. To this end, the majority confirmation is necessary in order to enable progress only among the active replicas in the majority partition. Reaching the event agreement in a minority partition is thus impossible.

We next propose two Event Agreement instances: a) the List-based Event Agreement (L-EA) and b) the Matrix-based Event Agreement (M-EA).

In Section 4.2.5 we provide a detailed analysis of the response time penalty imposed by the Event Agreement component. In this chapter, however, we limit our analysis to basic correctness evaluation using empirical validation.

3.4.4.1 List-based Event Agreement (L-EA)

With L-EA, reaching the agreement on the status of an observed replica requires collecting a repeated unbroken sequence of a minimum of $l_M \cdot (|C| - 1)$ matching observations from active replicas. l_M is the confirmation multiplier parameter allowing to suppress false positives created e.g., by repeated bursts of same events stemming from a single replica.

L-EA maintains a per-replica local and global counter of matching observations in the local and global failure (and recovery) lists of length $|C| - 1$, respectively. On suspicion of a failed replica, the observer replica increments the counter of the suspected replica and forwards its updated local failure

list to other active replicas. After receiving the updated list, the receiving replicas similarly update their own counter for any replica whose counter is set to a non-zero value in the received list. The active replicas continue to exchange and increment the local failure counter until eventually $l_M \cdot (|C| - 1)$ matching heartbeats are collected and thus the local agreement on replica's failed status is reached. The suspected replica's counter in the global failure list is then incremented as well, and process continues for the global agreement. If any active replicas identify the suspected replica as recovered, they reset the corresponding counter in both local and global lists and distribute the updated vector, forcing all other active replicas to similarly reset their counters for the suspected replica.

3.4.4.2 Matrix-based Event Agreement (M-EA)

Another realization of the Event Agreement entity is the matrix-based approach M-EA. Our design extends [196] to cater for the recovery of failed replicas and to feature the global agreement capability.

In summary, all replicas in M-EA maintain a status matrix, and periodically inform all other active replicas of their own status matrix view. The status matrix contains the vector with the locally perceived status for each observed replica, as well as the vectors corresponding to other replicas' views. Thus, each replica maintains its own view of the system state through interaction with local failure detector instance, but also collects and learns new and global information from other active replicas. The status matrix is a $|C|x|C|$ matrix with elements corresponding to a value from set {`ACTIVE`, `INACTIVE`, `RECOVERING`}. `RECOVERING` state is necessary in order to consistently incorporate a previously failed but recovered replica, so that all active replicas are aware of its reactivation.

Following a failure of a particular replica or a communication link to that replica, failure detector of an active neighboring replica initiates a trigger and starts suspecting the unreachable replica. The observing replica proceeds to mark the unreachable replica as `INACTIVE` in its locally perceived vector and subsequently informs all other replicas of the update in asynchronous manner. The remaining replicas store the state update in their own view matrix and evaluate the suspected replica for reached failure agreement. Agreement is reached when all active replicas have marked the suspected replica as `INACTIVE`.

In M-EA, each replica maintains two matrix instances:

1. *local* agreement matrix, where each locally perceived suspicion results in a state flip from `ACTIVE` → `INACTIVE`, and where a newly seen heartbeat for a previously failed replica leads to a state flip `INACTIVE` → `RECOVERING`. As soon as all active replicas have marked the suspected replica as `INACTIVE` (`RECOVERING` for a recovered replica), the local agreement has been reached (state flip `RECOVERING` → `ACTIVE` occurs for the recovered replica).

2. *global* agreement matrix, where state flip from `ACTIVE` → `INACTIVE`, and `INACTIVE` → `ACTIVE`, occurs only if the local agreement was previously reached for the updated state of the observed replica.

Dissemination triggers: Dissemination of the matrices is triggered periodically (according to the Signaling entity). If a replica has, however, observed a failure using its local failure detector

or has received a heartbeat from a replica considered inactive, it triggers the matrix dissemination asynchronously.

3.4.5 Evaluation and Results: Validation of Correct Raft Operation with Network Partitions

To demonstrate the correctness of the agreement-enabled consensus, we realize the connectivity graph depicted in Fig. 3.6 a) and gradually inject the link and replica failures as per Fig. 3.6 b). We first inject the three link failures at runtime. We then inject a failure in C2 and eventually recover it so to evaluate the correctness of both failure and recovery detection.

Fig. 3.7 depicts the three successive logical communication link failure injections and the subsequent replica failure injection in C2, as well as its recovery. The upper-left depicts the behavior of replicas during the link failure injections for replicas communicating using Best Effort Broadcast Dissemination (BEB-D) [49] (discussed in more detail in next chapter). After the missing signaling heartbeats were observed by the local failure detector in the affected replicas, failure suspicion is triggered for the unreachable replicas. Since six unidirectional link failures were injected (i.e., three bidirectional link failures), six local suspicions are triggered across the cluster (twice by C2 and once each by C1, C3, C4 and C5). In the case of Round Robin Gossip Dissemination (RRG-D) (ref. Section 4.2.3) depicted in lower-left, the local failure detector never triggers suspicions in any of the five controllers. This is due to propagation of heartbeats using gossip, where only a subset of all replicas must be directly reachable in order to disseminate the heartbeat consistently to all members.

The suspicions in the local failure detectors after individual link failures are insufficient to agree on the unreachable replicas as inactive, i.e., only direct partners on the failed link start suspecting unreachable replicas as inactive. The inactive replicas are correctly marked as such *only after* an actual failure in an actual failure in C2 (ref. center subfigure of 3.7). All active replicas eventually agree on C2's failure.

In the right-most subfigure, we recover C2 by restarting the replica and depict the time when the local and global agreement on its recovery are reached in the active replicas.

By means of the proposed mechanism, Raft's leader election can trigger only when global failure agreement is reached, thus solving the issues related to oscillating leader. The response time analysis, including detailed modeling of delays incurred by individual dissemination, signaling, failure detection and agreement components are given in Section 4.2.

3.5 Chapter Summary and Outlook

SDN enables the necessary control plane robustness by controller and state replication. However, the replication incurs the overhead in terms of resulting controller design complexity and response time / load penalty. It is not always obvious which particular cluster configuration would best suit the application and network configuration at hand.

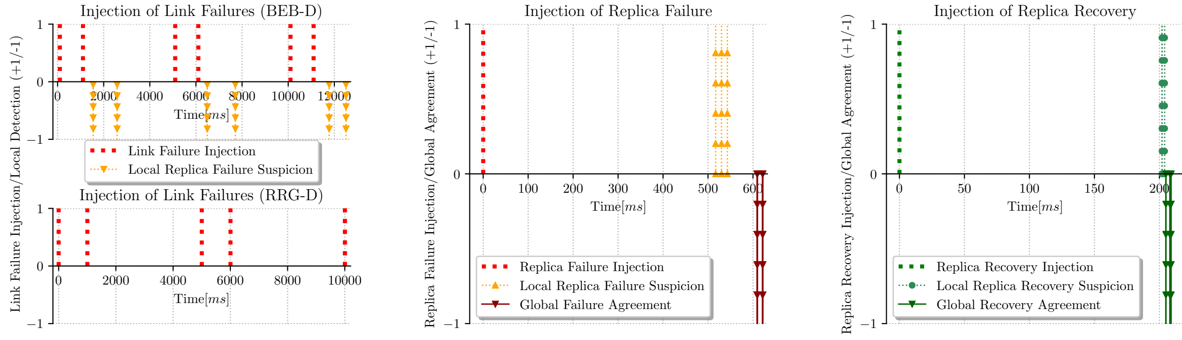


Figure 3.7: Link failure injections and the suspicions (left); the `C2` replica failure injection (center) and; replica `C2` recovery injection (right) for connectivity graph depicted in Fig. 3.6. While the broadcast-based failure detector (Best Effort Broadcast Dissemination (BEB-D)) tends to trigger local suspicion for disconnected replicas, gossip-based dissemination (Round Robin Gossip Dissemination (RRG-D)) ensures the heartbeats are eventually exchanged among all replicas, as long as no replica is *fully* partitioned away from the remainder of the network. The center subfigure depicts the time-points where the local and global agreement are reached among the active replicas. The right subfigure depicts the time-points where the local and eventually global agreement are reached for the recovered replica `C2`.

We have investigated the application of SANs for modeling and numerical evaluation of availability for arbitrary distributed cluster configurations. We have put special focus on the practically relevant distributed consensus algorithm Raft, but have generalized our model to be applicable to similar Paxos variants (e.g., Multi-Paxos). Indeed, Raft is implemented in two dominant open-source SDN platforms and is of practical relevance for performance analysis of the distributed control plane.

Using transient solver methods, we were able to provide an analytic estimation of the availability of a fault-tolerant control plane. We make similar guarantees for the response time system metric in the next chapter. Assuming a balanced distribution of controllers in the network w.r.t. the C2C delays, we conclude that larger control planes provide a higher system availability, eventually saturating at a certain threshold. In fact, assuming realistic failure rates and no long-term data plane partitions, 5 controller instances can guarantee a negligible unavailability of the control plane ($< 1e^{-9}$). To further improve the long-term control plane availability, we have proposed a watchdog mechanism for fast recovery from software failures and have proven its benefits on the long-term availability.

In the second part of the chapter, we have showcased the limitations of tightly coupled failure detection and consensus processes by reflecting on the example of non-reachable agreement in a Raft-based SDN controller cluster. In contrast to existing works, the proposed failure detection framework considers the possibility of limited knowledge of occurrence of network partitions in the controller replicas. We have solved the leader oscillation issue by introducing agreement as a necessary first step to confirming a particular replica's status, thus effectively ensuring no false positive failure / recovery detection ever arises, independent of the cluster size and type of the deployed failure detector.

We have validated the correctness of our design on an example of a problematic scenario and the corresponding framework implementation. We expect that this work motivates the future evaluations of distributed failure detectors in combination with consensus protocols as a set of loosely coupled but co-dependent modules.

In the next chapter, we investigate the response time characteristics of the presented Raft-based control plane and discuss its suitability for industrial applications requiring low response time.

Open points: Adaptations to proposed models to support the novel leaderless consensus protocol variants, such as EPaxos [132], require significant changes and are thus considered as future work. Additionally, the presented approaches are capable of tolerating Fail-Stop failures only and are thus indifferent to semantic- and aging-related system faults (i.e., generalized to Byzantine faults). In Chapter 6, we present strategies based on RSMs to enable tolerance against this class of faults.

Chapter 4

On Deterministic Response Time in Fail-Stop Control Plane

In the SC model, configuration requests facilitate a number of C2C synchronization steps prior to reaching consensus on a decision and finally executing the configuration change. For example, in an SDN module that subscribes to topology changes (e.g., raises alarms to an administrator in case of link failures), a topology change must first be committed across the cluster majority in the replicated Raft commit log, before the subscribed SDN module is notified and can execute its reactive step. The resulting response time is dependent on the the number and current availability of controllers, their execution performance and the C2C delays. As discussed in Section 2.2, clustered SDN controller solutions require estimation of their worst-case response times, before their deployment can be considered in critical infrastructure networks.

In the first part of the chapter we present the detailed SAN models for modeling worst-case control plane response times in a *partition-free* distributed control plane relying on Raft. In the second part, we additionally present a method for estimation of worst-case agreement delay for a failed / recovered controller replica in a Raft cluster *impacted by network partitions*. We assume a deployment extended with components for failure detection and Event Agreement component as presented in Section 3.4. We confirm the analytic formulations by empirical validation relying on varied cluster parametrizations and controller cluster sizes. Finally, we discuss the impact of each component of our design on the replica failure- and recovery-detection delay, as well as on the imposed communication overhead.

The proposed analysis methods and designs were published in augmented form in [8, 3].

4.1 Analytic Guarantees for Raft's Response Time

The final SAN model presented here details the distributed Raft commit operation, specifically the lifecycle of a Raft session during a client request handling procedure. By assuming reliable event delivery and bounded network, application and data store commit delays, we provide stochastic delay guarantees for response handling times in non-failure, partial-failure and cluster-majority failure states. As before, we consider the mechanism for fast recovery of controller bundles and processes (presented in Section 3.3.3) in evaluation of the resulting response time of the distributed control plane.

4.1.1 Related Work - On Predictable Response Time of Cluster Operations

Muqaddas et al. [135, 136] investigate the load overhead of the intra-cluster communication in a 2- and 3-controller ONOS cluster. They propose a model to quantify the traffic exchanged among the controllers and express it as a function of the network topology. They do not consider the effect of the transient failures on the response time and availability. Zhang et al. [199] describe the *single-data ownership* organizational model implemented by the Raft algorithm and propose an estimation formula for approximating flow setup time in a distributed SDN controller cluster. Their estimation is however fairly simplistic as it models only the average case. The worst-case estimations are not considered in their analysis.

Ongaro [223] and Howard et al. [87] provide initial performance evaluations of Raft. Howard et al. [87] furthermore implement an event-driven framework for prototyping of Raft using experimental topologies. Contrary to the analytical approach presented in our work, their performance evaluation of Raft is based on a limited number of repeated experiments and focuses on evaluating the Raft leader re-election procedure following a failure. Unfortunately, these works do not provide a good understanding of how the overall system response time is affected after a failure.

In two experiment-based studies, Suh et al. [181, 182] measure the throughput and the recovery time of a Raft-enabled SDN controller cluster with 1, 3 and 5 replicas. They put special focus on the effect of ϕ accrual failure detector [80] on the resulting performance footprint. The authors deduce that the controller fail-over time increases with ϕ. With higher ϕ, the ODL cluster becomes more conservative in determining a controller failure, hence in case of failures, using a large ϕ values will generally lead to slow failure discovery. Authors varied ϕ and measured the lowest recovery time of $\sim 2,6s$, which is a non-satisfying recovery time for many critical industrial applications. Instead of using an adaptive scaling factor ϕ, in the response analysis of the distributed Raft cluster, we rely on a fixed *follower timeout* variable with a mean of $\sim 225ms$. We assume that the C2C delays are bounded and will hence not exceed this value except in the case where a controller failure has occurred. This value is recommended by the authors of the Raft consensus algorithm, and was determined to be a good trade-off between recovery time and signaling overhead in their experiments [146].

Machida et al. [118] analyze the completion time of a job running on a server that is affected by *software aging*, and consider the benefit of the *preemptive-resume* operation, where a job resumes execution from the point of interruption as soon as the failed server recovers. Similar to this work, we investigate the job completion time for a client request, but we consider a distributed multi-server operation. We focus on the strategy where, assuming a failure occurs, the request is handled from the beginning instead of delegating it to the next server.

Paxos [111] is another influential [45, 48] consensus algorithm that originally motivated the design of Raft. Paxos ensures that any two distributed servers that are part of the same cluster may never disagree about the value of a particular update, for any applied update in the update history. In its optimized variations, its performance is comparable to Raft, in that, assuming a stable cluster leadership, committing a cluster-wide update takes a single round trip in most cases [133]. Multi-Paxos [218, 111] is a prominent variation of Paxos, that assumes a stable leader for an infinite number

of sequential cluster updates. This allows for one-round-trip delay as the first phase of Paxos becomes unnecessary for the majority of updates. In [214] the authors evaluate an implementation of Multi-Paxos and conclude that the overall performance of Multi-Paxos is limited by the slowest node in the fastest cluster majority. This is a valid observation for any quorum-based consensus algorithm, hence we distinguish the leader failures as critical for our analysis. Cheap Paxos [109] is another variation of Paxos which implements a dynamic cluster membership model, in order to allow making progress in face of up to F failures by deploying $F+1$ servers and F backup servers. Hence, Cheap Paxos involves less live-synchronized replicas but induces a non-negligible reconfiguration overhead after the replicas in the main system have failed [109].

To minimize the impact of SPOF and maximize the request load balancing, Mencius [121] proposes a round-robin-based handling of client requests by multiple leaders. Mencius indeed enables higher throughput in the stable case, however, the cluster continues to run at the speed of the slowest elected leader as new log updates may depend on actions assigned to the slow node. EPaxos [133] is a recent leader-less take on Paxos that tries to circumvent these issues. It keeps track of the ordering and mutual dependencies between the client-initiated updates. Hence, it is able to parallelize multiple update instances when no collisions between concurrent client updates are expected. Like Raft, it requires a single round-trip in most cases to commit a state update, and two round-trips if dependency conflicts arise. Contrary to Raft and other leader-based Paxos variations, the response time in an EPaxos cluster does not suffer from unstable leaders since the clients may always fall-back to any remaining live leader replica. However, it adds additional complexity in state-keeping and log compaction tasks because of the added dependency trees.

4.1.2 End-to-End Transaction Commit Model

The clients of the SDN control plane generate events, such as flow requests or switch notifications which necessitate data store updates in the controller and their subsequent synchronization across controller replicas. The client delivers these events in asynchronous and reliable manner to the replicas for processing. After the control plane finish the processing, the client is notified of the result. The SAN in Fig. 4.1 depicts this process.

Sub-process 1: The place `IdleState` models the initial system state where no events are queued for internal processing. Following a new event arrival at *any* of the Raft replicas, the receiving replica is tasked with the propagation of the new event to the current Raft leader. New event arrivals increment the token amount in the state `EventQueuedForLeader`, where events are queued until a leader becomes elected in the cluster. The input gates enumerated `LeaderAndMajorityUp#` ensure that the transmission of the event to the Raft leader or replicas, as well as the intermediate processing inside the cluster happens only in the case where both the Raft leader and the follower majority are active.

Sub-process 2: Propagation of the event from the furthest-away replica to the leader is modeled by the activity `delayToLeader` using a deterministic worst-case delay metric $T_R = T_{M_{worst}}$ (further discussed below).

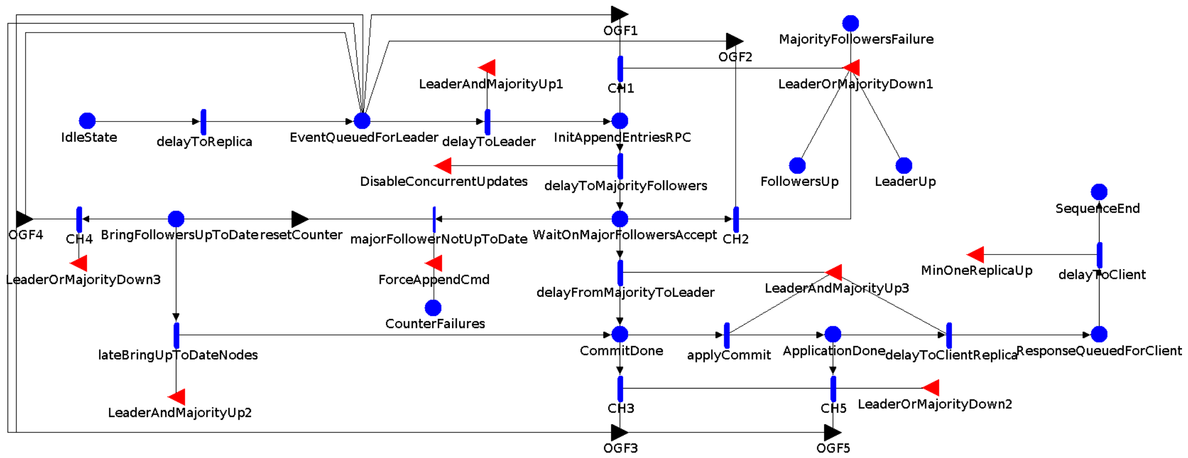


Figure 4.1: The Raft response time model depicting the sub-processes: (1) the reception of a client event at a follower proxy-replica; (2) the event propagation to the Raft leader; (3) the event propagation from the leader to the follower majority; (4) the data store commit of the client event; (5) its subsequent processing in the controller; and (6) the propagation of the response from leader, through the proxy-replica, to client.

Sub-process 3: On a received event, the leader initiates the propagation of the respective data store state update to its followers. The delays induced by the activities `delayToMajorityFollowers` and `delayFromMajorityToLeader` correspond, in the best-case to the leader-to-(preferred)-majority delay $T_{M_{best}}$. In the worst-case, to contact the follower majority, the leader needs to contact the follower furthest away from it and hence induce the worst-case uni-directional network delay $T_{M_{worst}}$. Thus, the delay between the Raft leader and the follower majority is governed by the number of failed followers.

Sub-process 4: When the follower majority has acknowledged the state update, the leader continues committing the data store change locally, and the system eventually reaches the `CommitDone` state. To prevent the leader from broadcasting multiple unacknowledged updates, we ensure the input gate `DisableConcurrentUpdates` enables the transition `delayToMajorityFollowers` if and only if the distribution states of Raft do not contain any outstanding tokens (no synchronization in progress).

Modeling Lagging Followers: In the case of at least one replica that necessarily comprises the cluster majority lags behind the Raft leader in terms of its *commit log* (ref. Section 3.2.5), the leader enforces additional steps in order to synchronize the cluster majority with its local view. To this end, the activity `majorFollowerNotUpToDate` fires and a token is incremented in the place `BringFollowersUpToDate`. The lagging followers are thus assigned the `Follower (Lagging)` node status (depicted in Fig. 3.2). For each Raft term state that is missing in the lagging follower, an additional round-trip delay in the critical path between the leader and replica is induced, hence adding $2 \cdot R_M \cdot T_M$ to the overall delay, where R_M is the maximum number of missing Raft terms in the follower. This delay is imposed in the definition of the activity `lateBringUpToDateNodes`. To govern the activation of the instantaneous activity `majorFollowerNotUpToDate` we make use of a stateful counter `CounterFailures` that is incremented on each new logged replica failure (ref. Section 3.3.2.1). We consider the worst-case and hence assume that at *any time* after $\left\lfloor \frac{|C|-1}{2} \right\rfloor + 1$ replica nodes have been disabled, out-of-date replicas are automatically present in the majority of the follower nodes required to confirm a leader update. Hence, we infer the additional overhead of updating the

lagging nodes in the replica majority. The flag to enable the activity `majorFollowerNotUpToDate` is cleared after the reconciliation (as a side configuration of the gate `resetCounter`).

Sub-process 5: Following an applied data store commit in the majority of replicas, the leader commits the state locally and the SDN application gets notified of the data store event. The data store commit and the SDN application's processing delay are induced during the activity `applyCommit` and are modeled as T_C and T_A in Table 3.1, respectively.

Sub-process 6: After the application has completed its processing (in `ApplicationDone`), the leader notifies the replica that initially generated the update event (thus adding once-more T_R to the overall worst-case delay), and the replica further forwards its response to the client (thus adding T_{CR} which is the client-replica delay). The system then finally reaches the stable state `SequenceEnd`, where the event is marked as successfully processed.

Modeling Reaction to Failure Events: In case of a failure occurrence in the leader or follower majority during the event processing, the activities named `CH#` lead to a token being shifted from the current SAN place to the `EventQueuedForLeader` place, using the output gate increment action modeled by `OGF#`. Hence, the event distribution procedure restarts as soon as the cluster is re-established. The delay until a critical failure occurrence of the leader or the follower majority is noticed by the client is modeled using the *client timeout* T_{CL}.

Modeling the delay from Raft leader to the furthest-away replica of the follower majority: This delay varies depending on the availability and proximity of followers w.r.t. the current cluster leader. We annotate the leader-to-majority followers delay as T_M. Assuming a deployment of $|C|$ controller nodes and a single leader L where $L \in C$ at any time, the set $S_L = \{D_{R_1}, D_{R_2} ... D_{R_{\lfloor |C|/2 \rfloor}}\}$ where $R_i \in C$ contains the maximum bounded delays between the leader L and $\lfloor |C|/2 \rfloor$ follower nodes *closest* to L w.r.t. delay between controller L and each of the *available* followers R_i. Hence, we define the delay between leader L and the follower majority as the delay between L and the *farthest* follower in the majority $T_M = max(S_L)$.

To emphasize the effect of a failed preferred-follower replica on the response time, in our exemplary evaluation, we scale the delay value to contact the followers majority linearly with the number of currently available followers using a scaling factor S_F so that:

$$T_M = \begin{cases} S_F \cdot T_{M_{best}} & \text{when } F_{up} \geq \left\lfloor \frac{|C|}{2} \right\rfloor \\ undefined & \text{otherwise} \end{cases}$$

For the evaluation purposes we model the S_F as a function of the current marking of SAN so that $S_F = \frac{|C|-1}{F_{up}}$ and thus:

$$T_M = \begin{cases} \frac{|C|-1}{F_{up}} \cdot T_{M_{best}} & \text{when } F_{up} \geq \left\lfloor \frac{|C|}{2} \right\rfloor \\ undefined & \text{otherwise} \end{cases}$$

where F_{up} represents the number of currently available followers. In the best case where all nodes are up, the leader-majority delay equals $T_{M_{best}}$. In the worst case, the controller-majority delay peaks at $T_{M_{worst}} = T_R = 2 \cdot T_{M_{best}}$ when only $\lfloor |C|/2 \rfloor + 1$ controller nodes (including the leader) are active.

Note: Using a fixed scaling factor is an exemplary and non-optimal representation, as the exact worst-case leader-majority delay is equal to the delay between the leader and the *farthest* away follower in the current follower majority, and hence necessitates knowing the exact bounded delays between each two SDN controllers in the network. We omit this level of model granularity as the required parameters would require population from an engineered network topology, and would further rely on the optimality of the used controller placement technique. Nevertheless, the SAN model proposed here can be extended to take an arbitrary set of C2C delay parameters with no added effort.

Data store sharding: The data store of an SDN controller (e.g., ODL) is sharded into an arbitrary amount of *data shards* at a flexible granularity (e.g., data shard per virtual topology). If all data shards are available on each replica, each available controller is an active member of each per-shard Raft session. Raft can then handle the updates of different data shards concurrently and in isolation. This enhances the overall throughput of the system as multiple asynchronous updates to different shards are parallelized and may execute without blocking.

4.1.3 Model parametrization using a Raft experiment

To validate the fitness of our response time model, we first compare the analytics solution against an experimental Raft setup in a zero-failure scenario. For this purpose, we implement a Raft agent and deploy multiple copies thereof in a Raft cluster. For the Raft backend implementation, we use the open-source Java library *libraft* [1]. The cluster is organized so that the controller nodes, acting as Raft agents, are reachable in an any-to-any manner over a single-hop Open vSwitch (OVS) [2] instance. The OVS is configured to inject a constant symmetrical delay of $5ms$ on each egress port. We use this value as a deterministic reference leader-to-follower-majority delay T_M in the model parametrization. Furthermore, based on the *raftlib* performance observations, we model the commit delay parameter T_A as an exponentially distributed delay with a mean of $1ms$. The resulting modeled response time and the comparison with the experimental results for different controller cluster sizes are depicted in Fig. 4.2. To reflect the stochastic performance guarantees when replica failures are concerned, we resort to using only SAN-based analytical modeling for most accurate approximations.

To evaluate the expected response time metrics of arbitrary cluster configurations, we vary the number of controller replicas that take part in the Raft session as per Table 3.1. The generalized long-term software and hardware failure rates, as well as the hardware repair rates are taken from Liu et al. [117]. As discussed in Section 3.3.2.1, to allow for granular worst-case response time analysis, we model single and correlated failure injections with varying number of failures, where following a failure, a replica is temporarily excluded from the cluster until recovered. To depict the benefits of failure source differentiation and the proposed Watchdog Mechanism (WD), we distinguish between

[1] libraft - Raft Distributed Consensus Protocol in Java: https://libraft.io
[2] Open vSwitch - Production Quality, Multilayer Open Virtual Switch: https://www.openvswitch.org/

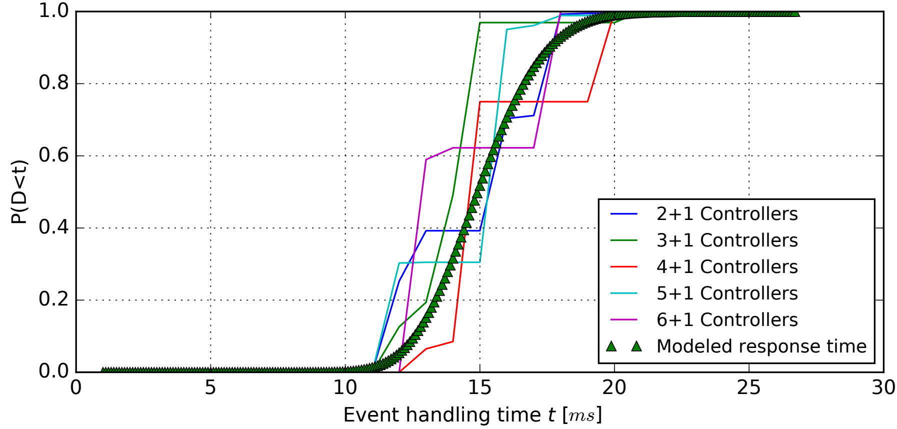


Figure 4.2: Comparison of experimentally observed and modeled Raft performance with clusters of various sizes. Represented are the Empirical Distribution Functions (ECDFs) of per-cluster-configuration measurements. Each measurement encompasses 1000 sequential write operations. The observed delay considers a fixed single-hop packet latency of $5ms$ in between the Raft leader and replicas, as well as a $1ms$ data store commit time in the leader and replica majority. Client and application delays were not considered in this experiment. The measurements were taken in a zero-failure state of the Raft cluster and should serve as an initial indicator of the response time model fitness.

mixed and software-only failures, and vary the number of failure injections between 1 and $\left\lfloor \frac{|C|}{2} \right\rfloor + 1$ (majority nodes down) controller failures.

The fact that the process of uniformization (ref. Section 3.3.1) may only be applied to exponentially distributed transition rates makes our estimations slightly pessimistic. Thus at the cost of the generated CTMC state size and required solving time, we approximate every deterministic message delay and timeout and minimize the total distribution variance using a 20-stage Erlang distribution.

4.1.4 Evaluation and Results: Response Time Analysis

When a single random-role controller from the SDN cluster fails as a result of a hardware, process or bundle failure (each being equally probable), deploying a larger number of controller replicas ensures an overall decrease in the expected response time (see Fig. 4.3). This is related to the probability of a leader being injected with a failure, hence necessitating a leader re-election to move forward the state. The probability of a leader failure becomes increasingly lower when larger clusters are deployed (as explained in Section 3.3.2.1).

Next, we evaluate the probability of meeting an event handling deadline when the majority of nodes in the cluster have failed. The expected response times where mixed hardware and software failures, as well as exclusively bundle-level failures may occur, are depicted with and without the WD enabled in Fig. 4.4a and 4.4b, respectively. The WD enables faster recovery of replicas and hence faster repeated processing of an event in the case of leader and follower majority failures. An SDN cluster equipped with the WD on average processes the events faster and with a higher probability than the one without. Especially when simultaneous hardware failures are improbable and software failures are typical, the fast software recovery provides obvious response-time benefits (Fig. 4.4b).

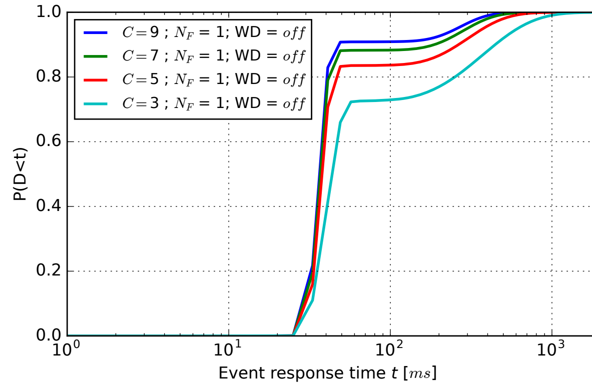
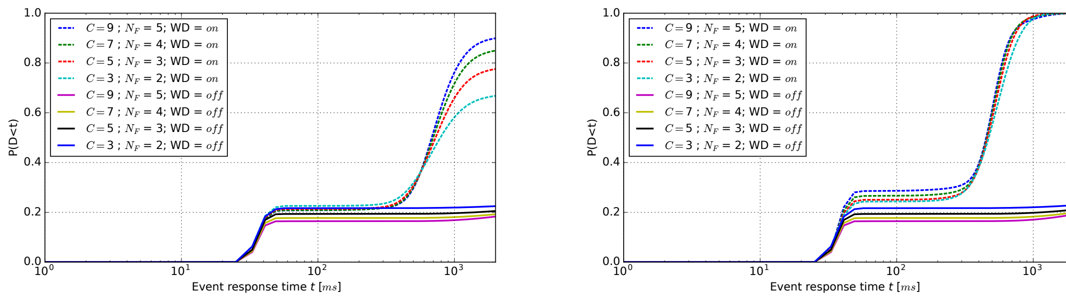


Figure 4.3: Varying probability of an event being successfully handled in a given time period t for different SDN controller cluster sizes $|C|$. The probability of the Raft leader failing is inversely proportional to the cluster size.



(a) Response time assuming an occurrence of N_F combined hardware and software (process, bundle) failures. All three types injections are equally probable.

(b) Response time assuming N_F bundle-only failures. The WD guarantees a timely repair and inclusion of the recovered replica in the cluster.

Figure 4.4: Probability of receiving an event response during an observation window, assuming a simultaneous occurrence of (a) N_F mixed and (b) N_F software-bundle only controller failures in a cluster comprised of $|C|$ controllers. The failures are injected at rate $N_F \cdot 0.0333$ (all N_F failures are thus expected to be injected by $t = 30ms$).

Fig. 4.5 depicts the effect of the consecutive failures on the experienced response time in a 7-node controller cluster. If the majority nodes remain available after each individual failure, the time to respond is governed by the case where a cluster leader fails and a new leader election procedure is automatically initiated. There is no noticeable difference in the convergence time regardless of the (non-)usage of the WD in this particular case. The lower the maximum number of induced failures induced, slightly shorter is the expected response time. This may be related to the fact that the follower timeouts are exponentially distributed, hence a higher number of active nodes that time out after a leader failure leads to an overall lower expected time to select a candidate and repair the cluster.

4.1.4.1 SAN Model Complexity and Solution Time

Compared to the manual modeling of Markov Chains, SANs allow for more compact modeling of complex scenarios. Analytically however, both options need to solve the same CTMC and have to deal with an exponential increase in model size which may result in inefficient or intractable analytical solutions when complex models are concerned [81, 161]. The model complexity dictates both the amount of computational resources and the time required to solve the model.

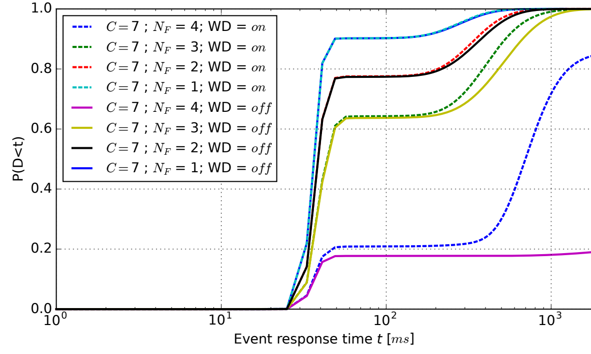


Figure 4.5: Resulting response time assuming an occurrence of $1 \leq N_F \leq (\lfloor\frac{|C|}{2}\rfloor + 1)$ controller failures in a 7-node controller cluster. The response time is governed by the duration of the leader election procedure. When the majority of controllers are unavailable, the usage of the WD (dashed) leads to important benefits w.r.t. expected worst-case response time.

Fig. 4.6 depicts the state space sizes of the generated CTMCs in our response time analysis. The generated state space is used by the transient solver to find the transient solutions for short-term (N_F lower than $|C|$) and long term (N_F considers up to $|C|$ failures) numerical analysis. The model complexity increases with the number of possible combinations the system may occupy. For short-term response time analysis we limit the complexity of the model by considering only the injected correlated failures - this is realistic as only a very short time period ($1s < x < 2s$) is considered (ref. Fig. 4.3 and Fig. 4.4). For long-term analysis, additional system states, where more than just the majority of nodes may fail could be of interest (consider Fig. 3.5). Fig. 4.6 shows the CTMC state space sizes for the cluster configurations up to $|C| = 19$. We observe that, for some parametrizations, the compiled state space size grows exponentially with the number of controller replicas. The number of possible failure injections dictates the number of generated unique combinations. For the most accurate setting of the $E_S = 20$ (20 Erlang stages, see Section 4.1.3) and cluster sizes of ≥ 17 replicas, we have encountered memory handling limitations in the flat state space generator in Möbius. Namely, if the solution should cover for all theoretically possible system combinations, i.e., when failure of every single node should be considered, the solution space eventually grows to an intractable amount of states for very large cluster sizes. To cater for the scalability of our solution when analyzing large control planes, we propose three options:

1. State space largeness avoidance by applying a scenario-based approach to the worst-case modeling. For example, one could consider a limited number of maximum failure injections. By limiting the number of maximum failure injections to $N_F = \lfloor\frac{|C|}{2}\rfloor + 1$, large-scale clusters can be analyzed successfully (ref. Fig. 4.6).

2. State space largeness avoidance by trading solution accuracy, e.g., by manipulation of the Erlang stages used for the approximation of the deterministic transitions.

3. Faster convergence of the transient solver by raising the error tolerance of the uniformization (ref. Section 3.3.1).

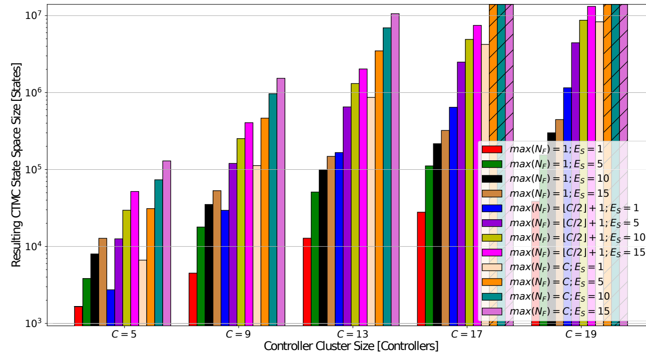


Figure 4.6: Size of the CTMC state space generated using the SAN models and parameters discussed in Chapters 3 and 4. The lower the number of controller failures of interest (i.e., for $N_F < |C|$), the smaller the resulting CTMC state space size. If the possibility of an eventual occurrence of failures in all nodes is assumed, the state space grows correspondingly, reaching up to 10^7 possible state space combinations with controller cluster size set to $|C| = 13$ and the maximum accuracy $E_S = 15$. Striped bars represent the unsuccessful CTMC compilations where the flat state space generator fails to compile the state space. However, by considering a lower number of Erlang approximation stages E_S, $|C| = 19$ and more controller replicas can be handled with a limited inaccuracy (ref. Fig. 4.7). Similarly, a focused assumption on the maximum number of possible failure occurrences helps the scalability of the solution (where $max(N_F) < |C|$).

For completeness, we also evaluate the second option by varying the Erlang stage parametrization. We take note of the effect on the overall result accuracy for the transient analysis of a 7-node cluster. Fig. 4.7 depicts the inaccuracy of the latency bound introduced by lowering the number of Erlang stages from $E_{S_{high}} = 20$ to $E_{S_{low}} \in \{5, 10, 15\}$. At $E_S = 5$, the generated state space size is decreased by a magnitude (see Fig. 4.6) and is hence, in addition to the first option, an effective method of deploying our models in a scalable manner. From this study, we conclude that the state space generation process is scalable as long as the accuracy and failure injection parameters are selected carefully for the use case at hand.

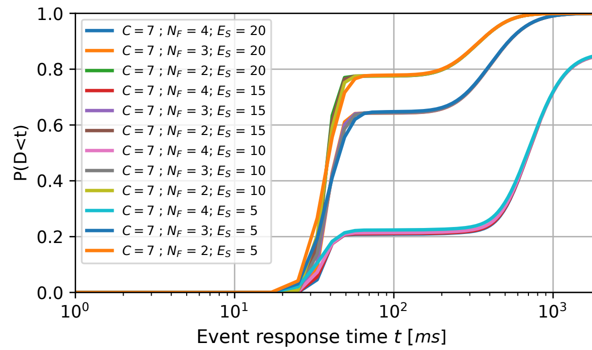


Figure 4.7: Inaccuracies stemming from a decreased number of Erlang stages E_S used in the approximation of deterministic transitions are negligible. Inaccurate approximation of a deterministic distribution lead to a higher variance for the random variable describing the failure arrivals. Hence, for small E_S the solver estimates a more relaxed (thus more pessimistic) latency bound.

We next consider the performance overhead of the state space generation in our approach. Fig. 4.8 shows how the scenario where $max(N_F) = |C|$ with $|C| = 19$ and $E_S = 5$ results in a tolerable $\sim 10^3$ seconds solving period.

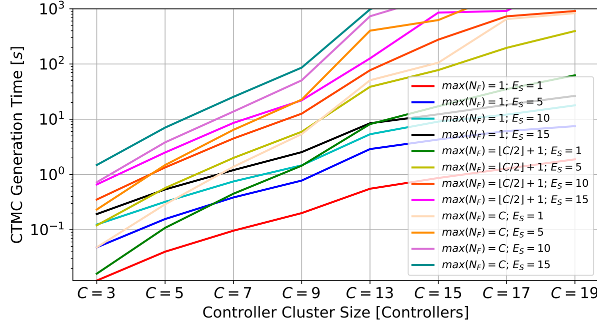


Figure 4.8: Overhead of the CTMC compilation for varying cluster sizes $|C|$, failure injection counts $max(N_F)$ and Erlang parametrizations E_S. While very accurate and large-scale combinations may lead to intractable solutions, feasible solutions can be presented even for the complex deployments of $|C| = 19$ controllers with various degrees of accuracy and all $max(N_F)$ combinations.

Fig. 4.9 depicts the computation time to solve the presented SANs. The duration of the solution computation of SAN will vary depending on the model complexity (state space size), the definition of the observed performance variable (reward function and the number and granularity of time measurements), as well as the required accuracy and model *stiffness* (the range of the expected action completion times) [120]. In the Möbius modeling tool, the accuracy of the transient solver indicates the degree of accuracy that the user wishes in terms of the number of decimal places. The solver execution times depicted in Fig. 4.9 were observed for the accuracy parameter set to 9 and an observation window of $1s$ (1000 data points). The largest generated state space for the purpose of modeling the largest cluster size necessarily leads to the longest solution computation times. For the analysis scenarios described here, these computation times are feasible.

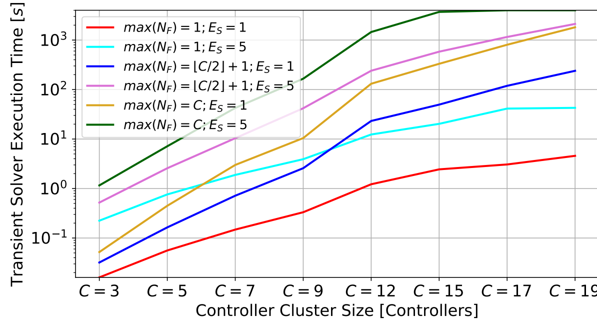


Figure 4.9: Computation time of the *instant of time* [55] transient solutions for the state space sizes depicted in Fig. 4.6. The solution covers the target observation interval of 1 second at millisecond resolution - hence the transient solver has computed the solutions for 1000 time-points. The computations were executed on a commodity hardware equipped with a modern Advanced Micro Devices (AMD) CPU and 32GB of DDR4 memory. The required computation overhead for the numerical solution is feasible for a short-term response time study.

4.2 Upper Time Bounds on Distributed Failure Detection

In Section 3.4, we solved the issue of oscillating consensus leadership and unmet consensus by disaggregating the process of distributed agreement on the failure event, from the underlying failure

detection. We concluded that we require agreement among the active controller replicas on the status of the observed cluster member, prior to reacting to the status update (i.e., by re-initiating leader election).

In the remainder of the chapter, we break down the design of the resulting framework into four base components: signaling, dissemination, event agreement (ref. Section 3.4.4), and failure detection. We introduce two exemplary instances of each entity and provide an analytical model for computation of the worst-case period necessary to reach agreement on replica status. In the evaluation part, we evaluate the developed framework in all possible coupling variants using varied parametrizations. We then discuss their performance trade-offs and summarize the co-dependency effects using metrics of event detection time and recovery and communication overhead. We confirm the correctness of the analytical formulation for the worst-case agreement period on a replica's status using empirical measurements.

4.2.1 Related Work - On Deterministic Convergence Time in Face of Network Partitions

Hayashibara et al. [80] introduce the ϕ-based Failure Detector (ϕ-FD). While similar in performance to other well-known failure detectors at the time [40, 191], ϕ-FD was shown to provide a greater tolerance against large delay variance in wide-area networks. A minor variant of ϕ-FD using a threshold-based metric for higher tolerance against message loss was proposed in [166]. ODL [123] uses Akka Remoting and Clustering [3] for remote replica operation execution as well as its failure detection service. Akka implements the ϕ-FD [80], hence we focus on the original ϕ-FD variant in this work as well. Amazon Dynamo [58] is another failure detector type that relies on randomized pinging and distributed failure detection. It is not investigated here as it cannot guarantee the property of strong completeness in bounded time.

Yang et al. [196] motivate the usage of a matrix-based approach for reaching agreement on SDN controller failures. The authors base their evaluation on a binary round robin gossip variation, originally proposed in [155]. We reproduce the same gossip variation in our evaluation as it allows for deterministic estimation of the worst-case convergence time. In contrast to [196], however, we also vary the coupling of other components of the event detection framework, evaluate the framework in failure scenarios and extend the matrix-approach to support global event agreement and consistent replica recovery. Katti et al. [101] introduce a variation of a list-based detector in order to decrease the footprint of matrix-based agreement. They, however, do not provide for a method to converge on replica recovery nor do they provide for analytical bounds on the expected system performance.

In [158], Van Renesse et al. propose one of the earliest failure detectors using gossip dissemination. In [157], the authors propose a flow control mechanism for gossip-based anti-entropy reconciliation to ensure fairness among the members actively gossiping the updates. Katti et al. [101] propose and evaluate a list-based agreement algorithm with random-destination gossip dissemination technique.

[3] Akka - toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala - https://akka.io/

However, their approach cannot guarantee the upper bound of number of gossip rounds required to converge the same update in all replicas.

Suh et al. [181, 182] evaluate the Raft leader election time following ODL leader failures. The authors do not consider data-plane (i.e., link / switch) failures or partition occurrence between the cluster members. Similarly, only the built-in variant of ODL's failure detection (with its non-gossip dissemination) is considered in both works.

Implementations of consensus algorithms in networking hardware, e.g., those of Paxos [212, 57] and Raft [200]) have recently started gaining traction. Dang et al. [212, 57] portray throughput, latency and flexibility benefits of network-supported consensus execution at line speed. We expect to observe similar advantages in event detection time if our framework was to be realized in network accelerators and programmable P4 [46] switches but leave this topic for future investigation.

4.2.2 Reference Model of Decoupled Failure Detection

The abstract reference model depicted in Fig. 4.10 portrays the interactions among the proposed core entities:

- *Failure Detection (*-FD)*: Triggers the suspicion of a failed replica by means of local observation - e.g., an *active* replica may suspect a remote replica as *inactive* following a missing acquisition of replica's heartbeats.
- *Event Agreement (*-EA)*: Deduces the suspected replica as *inactive* or recovered following the collection and evaluation of matching confirmations from at least $\lceil \frac{|C|+1}{2} \rceil$ *active* replicas. It uses the underlying Failure Detection to update its local view of other replicas' state.
- *Signaling (*-S)*: Dictates the semantics of the interaction between processes, e.g., the protocol (ping-reply or periodic heartbeat exchange), and configurable properties (e.g., periodicity of view exchange). Signaling ensures that the local view of one replica's observations is periodically advertised to all other reachable replicas.
- *Dissemination (*-D)*: Dictates the communication spread (e.g., broadcast / unicast or relay using gossiping). As depicted in Fig. 4.10, the Dissemination is leveraged for periodic signaling of replica's status view (by the Signaling entity), as well as for triggering asynchronous updates on newly discovered failures (by the Failure Detection).

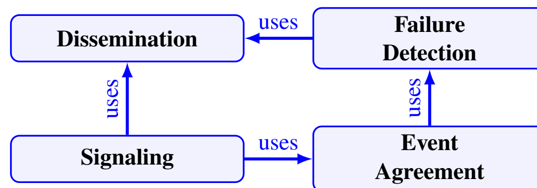


Figure 4.10: The proposed event detection reference model.

We next identify two exemplary instances for each of the four entities and provide an analytic expression for the worst-case delay, specific to that instance. The sum of the delays introduced by each of the components denotes the predicted waiting period to reach agreement on an observed remote replica's status. Note that the worst-case convergence time can be determined only in scenarios where all active replicas can communicate either directly or through relay replicas.

4.2.3 Modeling Dissemination and Signaling Delays

In our model: i) replicas learn about the status of other members (active or inactive) using local Failure Detection; and ii) confirm those assumptions by collecting information from active remote members in the Event Agreement module. Dissemination governs how the information containing replicas' local view of other cluster members' state is exchanged. We exemplarily distinguish Round Robin Gossip Dissemination (RRG-D) and Best Effort Broadcast Dissemination (BEB-D) dissemination.

4.2.3.1 Round Robin Gossip (RRG-D) Dissemination

In RRG-D, message transmissions occur periodically on a per-round basis. During each round, the gossip sender selects the receiver based on the current round number. This ensures that in a fault-free setup, given periodic heartbeat message exchange, each state view update is propagated to all cluster members in a maximum of $\lceil log_2(|C|) \rceil$ subsequent rounds. The heartbeat destination identifier ID_{Dst} is selected based on current round R_i as:

$$ID_{Dst} = ID_{Src} + 2^{R_i - 1} \mod |C| : 1 \leq R_i \leq \lceil log_2(|C|) \rceil$$

Non-Failure case: The worst case incurred by the gossip transmission between i and j in the *non-failure case* corresponds to the sum of delays on the longest directional gossip path (but limited by $log_2(|C|)$) and the signaling interval T_S:

$$T_D(i,j) = \sum_{l=1}^{log_2(|C|)} (h_l + T_S) : h_l \in \bigcup_{k=1}^{log_2(|C|)} \mathcal{H}_k^{i,j}$$

where set $\mathcal{H}_k^{i,j}$ contains the k-th largest element of delay set $\mathcal{D}^{i,j} \subseteq \mathcal{D}$ (union of delays for all links of all unidirectional gossip paths between i and j), determined using induction:

$$\mathcal{H}_k^{i,j} = \{a \in \mathcal{D}_{k-1}^{i,j}; a \geq b \forall b \in \mathcal{D}_{k-1}^{i,j}\}$$
$$\text{with } \mathcal{D}_0^{i,j} = \mathcal{D}^{i,j}; \mathcal{D}_k^{i,j} = \mathcal{D}_{k-1}^{i,j} \setminus \mathcal{H}_k^{i,j}$$

Failure case: For the *failure case*, we assume an availability of only one path between the replicas i and j, hence the worst-case dissemination delay corresponds to the sum of all delays across all replica pairs on the longest gossip path $\mathcal{P}_{i,j}$ and the periodic signaling interval T_S:

$$T_D(i,j) = \sum_{(k,l) \in \mathcal{P}_{i,j}} (d_{k,l} + T_S) : d_{k,l} \in \mathcal{D}$$

4.2.3.2 Best Effort Broadcast (BEB-D) Dissemination

Heartbeats in BEB-D are propagated from source replica to all remaining cluster members concurrently in a single round. In contrast to RRG-D, in the *non-failure case* messages must not be relayed, and each message transmission incurs an overhead of $O(|C|-1)$ concurrent transmissions. The worst case delivery time between replicas i and j in the *non-failure case* corresponds to the sum of the worst-case uni-directional delay $d_{i,j}$ and the signaling period T_S. In the *failure case*, intermediate replicas must relay the message, hence the worst-case equals the gossip variant:

$$T_D(i,j) = \begin{cases} d_{i,j} + T_S : d_{i,j} \in \mathcal{D} & \text{if } \textit{non-failure case} \\ \sum_{(k,l) \in \mathcal{P}_{i,j}} (d_{k,l} + T_S) : d_{k,l} \in \mathcal{D} & \text{if } \textit{failure case} \end{cases}$$

We leverage the unidirectional heartbeats and ping-replies as carriers for transmission of the Event Agreement payloads.

4.2.3.3 Uni-Directional Heartbeat (UH-S) Signaling

With Uni-Directional Heartbeat Signaling (UH-S), communicating controllers advertise their state to the cluster members in periodic intervals. The periodic messages are consumed by the failure detectors (ϕ-FD and Timeout-based Failure Detector (T-FD), ref. Section 4.2.4) to update the liveness status for an observed replica. The parametrizable interval t_S denotes the per-destination round duration between transmissions.

4.2.3.4 Ping-Reply (PR-S) Signaling

With Ping-Reply Signaling (PR-S) signaling, transmissions by the sender are followed by the destination's reply message containing any concurrently applied updates to the destination's Event Agreement views (i.e., the *local* or *global* agreement matrix or the agreement list).

The imposed worst-case waiting time for both UH-S and PR-S equals the configurable waiting period between transmissions $T_S = t_S$.

4.2.4 Modeling Failure Detectors Delay

We next outline the worst-case waiting time for a successful local failure detection of an inactive replica using the ϕ-based Failure Detector (ϕ-FD) [80] and the Timeout-based Failure Detector (T-FD).

4.2.4.1 φ-Accrual Failure Detector (φ-FD)

To reliably detect a failure, the ϕ-FD must detect a suspicion confidence value higher than the configurable threshold ϕ. The function $\varphi(\Delta_T)$ is used for computation of the confidence value. To guarantee the trigger, it must hence hold:

$$\varphi(\Delta_T) \geq \phi$$

where Δ_T represents the time difference between last observed heartbeat arrival and the local system time. The accrual failure detection $\varphi(\Delta_T)$ leverages the probability that a future heartbeat will arrive later than Δ_T, given a window of observations of size ϕ_W containing the previous inter-arrival times:

$$\varphi(\Delta_T) = -log_{10}(P'_{\mu,\sigma}(\Delta_T))$$

Assuming normally distributed inter-arrivals for previous inter-arrival observations, $P'_{\mu,\sigma}(\Delta_T)$ is the complementary Cumulative Distribution Function (CDF) of a normal distribution with mean μ and standard deviation σ. We are particularly interested in the earliest time difference Δ_T^F at which the failure suspicion is triggered, i.e., the time difference for which it holds $\varphi(\Delta_T^F) = \phi$. From here we can directly expand the expression to:

$$\begin{aligned}
\varphi(\Delta_T^F) &= \phi \\
-log_{10}(P'_{\mu,\sigma}(\Delta_T^F)) &= \phi \\
-log_{10}(1 - P_{\mu,\sigma}(\Delta_T^F)) &= \phi \\
1 - P_{\mu,\sigma}(\Delta_T^F) &= 10^{-\phi} \\
1 - \frac{1}{2}(1 + erf(\frac{\Delta_T^F - \mu}{\sqrt{2} \cdot \sigma})) &= 10^{-\phi} \\
erf(\frac{\Delta_T^F - \mu}{\sqrt{2} \cdot \sigma}) &= 1 - 2 \cdot 10^{-\phi}
\end{aligned}$$

where $erf()$ represents the error function and $erfinv()$ is its inverse [69].

Resolving after Δ_T^F evaluates to:

$$\Delta_T^F = \sqrt{2} \cdot \sigma \cdot erfinv(1 - 2 \cdot 10^{-\phi}) + \mu \tag{4.1}$$

Note, however, that the recalculation of $\varphi(\cdot)$ executes in discrete intervals. Thus, to estimate the worst-case waiting period after which ϕ-FD triggers, we must also include the configurable recalculation interval φ_R:

$$T_{FD} = \Delta_T^F + \varphi_R$$

4.2.4.2 Timeout-based Failure Detector (T-FD)

T-FD triggers a failure detection after a timeout period t_T has expired, without incoming heartbeats transmissions for the observed replica. Compared to the accrual ϕ-FD detector, it is less reliable and prone to false positives in the case of highly-varying network delays. The worst-case waiting period introduced by T-FD corresponds to the longest tolerable period without incoming heartbeats:

$$T_{FD} = t_T$$

where t_T is the configurable fixed timeout interval. In contrast to ϕ-FD, its configuration parameter is intuitive to select. Furthermore, its analytical solution does not require collection of current samples for μ and σ estimation (ref. Eq. 4.1).

4.2.5 Modeling Event Agreement Delay

As discussed in Section 3.4.4, the Event Agreement component guarantees that all active replicas have eventually reached the same conclusion regarding the status of the observed replica. We next formulate the resulting delay in agreement, incurred by the two introduced instance types.

4.2.5.1 List-based Event Agreement (L-EA)

The induced worst-case delay in achieving the global agreement on the state of a monitored replica can be expressed (after simplification) as:

$$T_C = 2 \cdot l_M \cdot O(C) \cdot (\max_{i,j \in C} T_D(i,j) + 1)$$

where $\max_{i,j \in C} T_D(i,j)$ corresponds to the worst-case dissemination time between any two active replicas i and j and C is the computational overhead of list processing time in the source and destination replicas (merger and update trigger).

4.2.5.2 Matrix-based Event Agreement (M-EA)

Assuming confirmation of a failure detection by each active replica, the worst-case duration for global agreement equals the time taken to exchange the perceived failure updates and reach global and local agreement on the target's state:

$$T_C = 4 \cdot (\max_{i,j \in C} T_D(i,j) + O(C))$$

where $\max_{i,j \in C} T_D(i,j)$ corresponds to the worst-case dissemination time between any two active replicas i and j and C is the computational overhead of matrix processing time in the source and destination replicas (merger and update trigger). Two rounds (two unidirectional dissemination delays)

are required to synchronize the update in the local agreement matrix between: i) the replica that most-recently lost the connection to the failed replica; and ii) the most remote other active replica. Correspondingly, global agreement matrix views are exchanged only after the local agreement is reached, thus adding additional two delay rounds to T_C (total of four).

4.2.5.3 Worst-Case Convergence Time

Upper bound event detection convergence time corresponds to the sum of the time taken to detect the failure and time required to reach the global agreement across all active replicas:

$$T_{WC} = T_C + T_{FD} \tag{4.2}$$

T_C and T_{FD} are both functions of T_D, thus signifying the importance of evaluation of the performance impact in decoupled manner. In empirical evaluation in Section 4.2.6, we conclude that the presented worst-case analysis is pessimistic and may be hardly reachable in practice. Hence, we also highlight the importance of evaluation of the average case in experimental evaluation of different combinations.

4.2.6 Evaluation and Results: Convergence Time of the Distributed Failure Detection

To evaluate the impact of different instances of the four components of our event detection framework, we implement and inter-connect each combination as a set of loosely coupled Java modules. We vary the configurations of particular instances as per Table 4.1 so to analyze the impact of parametrizations.

Parameter	Intensity	Unit	Meaning	Entity Instance
$\lvert C\rvert$	$[4, 6, 8, 10]$	N/A	No. of controller replicas	ALL
l_M	$[2, 3]$	N/A	Confirmation multiplier	L-EA
t_S	$[100, 150, 200]$	[ms]	Signaling interval	UH-S, PR-S
t_T	$[500, 750, 1000]$	[ms]	Timeout threshold	T-FD
ϕ	$[10, 15, 20]$	N/A	Suspicion threshold	ϕ-FD
φ_W	$[1000, 1500, 2000]$	N/A	Window Size of Inter-arrival Time Observations	ϕ-FD
φ_R	$[100, 150, 200]$	[ms]	ϕ Recalculation time	ϕ-FD
$O(C)$	1	[ms]	Processing overhead constant	L-EA, M-EA

Table 4.1: Parameters used in evaluation of M-EA, L-EA, T-FD, ϕ-FD, RRG-D, BEB-D, UH-S and PR-S.

We inject a failure in a randomly selected replica, and subsequently measure the time required to reach the local and global agreement on the injected failure. After a fixed period, we recover the failed replica so to measure the necessary time to reach the agreement on recovery. Here, we omit link failures so to measure the raw event detection performance in average case.

We repeat the measurements 20 times for each of the 2^4 couplings, and for each parametrization extract the following metrics:

1. empirically measured time to reach the *local agreement* on a remote replica's *failure*;
2. empirically measured time to reach the *global agreement* on a remote replica's *failure* and *recovery*;

3. the *average* communication overhead per replica; and

4. analytical worst-case failure detection time for observed parametrization (per Eq. 4.2).

We have used `iptables` to inject communication link failures at runtime (i.e., by forcefully blocking communication between individual controllers) and `cpuset` to attach the processes to dedicated CPU cores. Replica failures were enforced by sending `SIGKILL` signal to the targeted process.

4.2.6.1 Impact of Failure Detection Algorithm Selection

Fig. 4.11 portrays the performance of the adaptive ϕ-FD and the timeout-based T-FD failure detectors. We have evaluated ϕ-FD with varying observation window sizes φ_W and suspicion thresholds ϕ, but did not observe important gaps compared to the presented cases. The depicted ϕ-FD parametrization corresponds to $\varphi_R = 150ms$, $\varphi_W = 1500$, and $\phi = 15$.

For T-FD, we varied the timeout threshold t_T. We observe that for networks inducing transmission delays with little variation, T-FD provides similar performance as ϕ-FD, given a low t_T threshold (i.e., the $t_T = 500ms$ case). For more relaxed parametrizations of t_T, active processes take longer to reach agreement on the updated replica status. Hence, failure detection agreement time of T-FD is proportional to t_T. The performance of both ϕ-FD and T-FD suffers for larger clusters.

We note that the advantage of ϕ-FD lies in its adaptability for networks with large delay variations, as it suppresses false positive detection better than T-FD does. The communication overhead, as well as the time to reach agreement on replica's recovery were not influenced by the failure detector.

Theoretical worst-case agreement time bounds for detecting replica failures (ref. Eq. 4.2) are depicted as horizontal black lines for each corresponding configuration in the upper-right Fig. 4.11. The measured empirical worst-case delay estimations have always stayed below this bound, thus implying the correctness but also the pessimism of the analytic approach.

4.2.6.2 Impact of Event Agreement Method Selection

We next evaluate the matrix-based M-EA and the list-based L-EA event agreement. The results are depicted in Fig. 4.12. We vary all other parameters apart from those of Event Agreement instances and aggregate the results in the box-plots, hence the relatively large variability in distributions.

M-EA has the advantage of faster agreement on a replica's failure and recovery. The increase of l_M multiplier from 2 to 3 showcases the importance of correct parametrization of the selected agreement instance. With the higher multiplier, the probability of detecting false positives with L-EA decreases, at expense of requiring a higher number of matching messages to confirm the status of an observed replica. In contrast, M-EA does not have this drawback as its agreement requires a single confirmation from other replicas.

Notably, we observe that L-EA converges faster to the agreement with the higher number of deployed controllers, but only if BEB-D is used as Dissemination method. M-EA's performance decreases with larger cluster sizes.

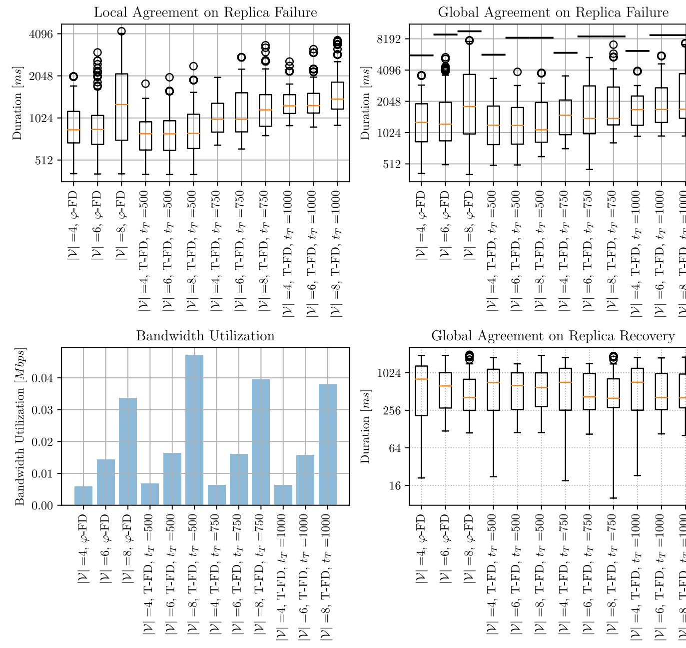


Figure 4.11: Impact of selection of the Failure Detection method. T-FD's performance is inversely proportional to the timeout threshold t_T, and given a low t_T, it provides similar performance as ϕ-FD accrual detector. However, ϕ-FD is better suited for networks with control channels of high variance latency.

The payload size of matrix view grows quadratically with controller cluster size. Compared to L-EA, this makes M-EA less efficient in terms of communication overhead, as can be confirmed by the per-controller loads depicted in Fig. 4.12.

4.2.6.3 Impact of Signaling and Dissemination Method Selection

We next vary the signaling methods and the corresponding heartbeat inter-arrival times. We have observed no critical advantage of the PR-S over the periodic UH-S design. In fact, PR-S comes at the expense of a relatively large communication overhead, as bidirectional confirmations are transmitted on each transmitted heartbeat. For both PR-S and UH-S, we note that the heartbeat periodicity largely impacts the required time to discover and reach agreement on the replica failure and recovery. The highest sending rate (at $100ms$ frequency) results in the best performance for both above metrics, but clearly comes at expense of the largest communication overhead.

Fig. 4.13 compares the implications of using either BEB-D or RRG-D as the Dissemination method. BEB-D ensures faster agreement for inactive replica discovery, at expense of a larger bandwidth utilization. This is due to the design of RRG-D that propagates the messages in a round-wise manner, thus strictly guaranteeing the convergence time of replicas' states only after the execution $\lceil log_2(|C|) \rceil$ rounds (in the *non-failure case* case). With BEB-D, however, the time required to agree on replica status scales better with the higher number of controllers.

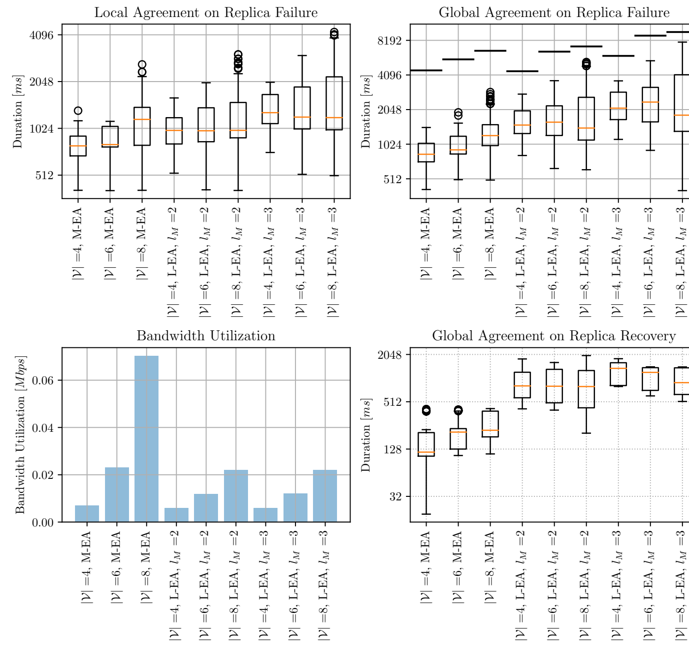


Figure 4.12: Impact of selection of the Event Agreement method. L-EA takes longer to converge than M-EA, both for the replica failure and recovery agreement time. L-EA, however, offers a lower total communication overhead, as well as a smaller computation and memory footprint. Its detection performance scales inversely proportional with the l_M multiplier. None of the depicted combinations have resulted in false positives, hence the lower values of l_M are certainly practical and can be further tuned.

4.3 Chapter Summary and Outlook

We have shown that, assuming a balanced distribution of controllers in the network w.r.t. the C2C delays, *worst-case* response time in a distributed cluster grows in inverse proportion to the control plane size. Using the proposed analytical modeling, determining the best suited cluster parametrization for a target network becomes possible without costly hardware setups for experimental evaluation or lengthy simulation runs. Analytical modeling further provides for corner case coverage and tighter stochastic guarantees than possible using simulations or experimental sampling.

Using transient solvers, we have proven the benefits of fast rejuvenation on the short-term response time. The solutions to our models are computationally feasible for both clusters optimized for extremely high availability (3-5 controllers), and complex clusters that may implement e.g., a *sharding*-based [182] design for highly scalable operation (up to $\sim$ 20 controllers).

In fact, assuming single-controller failures and a proper tuning of Raft parameters (i.e., setting low candidate / follower timeouts), strict response time requirements of $\leq 500ms$ for TSN control planes can indeed be provably met by Raft-based SDN controller clusters. Scenarios requiring zero fail-over time and Byzantine fault-tolerance, however, necessitate design extensions, as shown in later chapters of this thesis.

In the second part of the chapter we have investigated the delay incurred in agreement on a failed / recovered replica's status using our failure detection framework. The proposed framework considers

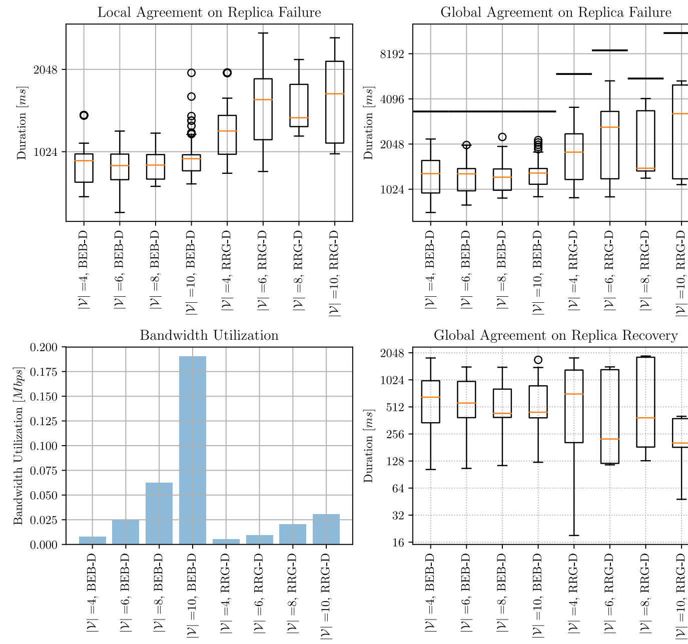


Figure 4.13: Impact of selection of the Dissemination method. The agreement time of gossip-enabled RRG-D lacks compared to the BEB-D broadcast-based dissemination. However, the gossip approach comes with the benefit of a smaller communication overhead. BEB-D scales better with the higher number of cluster members, due to the direct relation between the number of rounds required rounds to converge on an update with RRG-D and the size of the cluster.

the possibility of limited knowledge of occurrence of network partitions in the controller replicas. We have highlighted the impact of two instances of each sub-component of the model by means of exhaustive parametrization and varied cluster sizes. We have shown that the isolated implementation of each of the discussed components has a notable impact on the expected detection time and the communication overhead of the overall solution. Finally, we have discussed the analytic formulation for the worst-case failure detection time and have demonstrated its correctness by empirical validation.

Open points: We provide the response time metrics for a model that assumes an accumulated distribution of Raft state updates in the latency- and throughput-optimized, batched mode. Extending the proposed model to support a sequential distribution of updates at high scale is non-trivial using SAN-based modeling, because of the added state size complexity. Supporting the queuing behavior when handling client-generated events would require inclusion of additional formalisms e.g., from Queuing Petri Nets and is considered as future work.

Likewise, we leave the investigation of possible benefits of hardware-accelerated (e.g., P4-based [46]) distributed failure detection framework for future research. Exposing event detection as a global service, as well as enabling faster convergence times (e.g., matrix- / list-based merge procedure in hardware) could lead to a better performing detection and lowered overhead in the end-host applications, compared to the current model where each application implements its event detection service independently.

Chapter 5

Lowering Response Time in Fail-Stop Control Plane

We distinguish two typically deployed models for synchronizing state across a distributed SDN control plane. The Strong Consistency (SC) model [87, 218, 149] requires that the distributed state across cluster members is replicated and, following any single state-update at state leader, propagated using mutual consensus to replicas. In contrast, the Eventual Consistency (EC) model [35, 36, 113] omits the consensus procedure and guarantees that at least one delivery invariant holds. However, the advantage of non-blocking operations comes at the expense of sacrificing the total ordering of state-updates and sometimes the system correctness.

In this chapter, we investigate the advantages and drawbacks of strong and eventually consistent state replication on the performance and the resulting response time of the distributed control plane. Specifically, we consider the impact of the deployed consistency model on scalability (i.e., scaling of response time with control plane size) and correctness metrics of the control plane applications. We evaluate the performance of SC and EC models, and make a case for the novel Adaptive Consistency (AC) design. We show how, in contrast to the SC model, the proposed AC model offers additional benefits w.r.t. request-handling throughput and response time, while guaranteeing correctness semantics, that are unachievable with the EC paradigm.

The proposed analysis methods and designs were published in augmented form in [2, 10]. *Additionally, interested readers are referred to our publication* [12], *which introduces* `SEER` *- a data-driven approach to performance prediction and efficient leader election with goal of response time minimization in SC* SDN *control plane. Due to space constraints,* [12] *is not discussed in this chapter.*

5.1 Background on Different Consistency Models

With the concept of state replication, the SDN controller instances replicate their data store contents to other members that take part in a logical *controller cluster*, using a state distribution protocol of choice (e.g., Raft [3, 87]). When a failure of a controller is suspected, another replica from its *cluster* is able to take over and continue to serve future application's requests. The selection of the *consistency*

model leveraged by the replication process affects the incurred synchronization overhead in terms of the resulting packet load, the experienced commit response times and the processing order of commits.

Strong Consistency (SC): In SC, each consecutive operation that modifies the internal state of the controller is serialized and confirmed by a quorum of replicas, before forwarding the state and processing subsequent transactions. In the leader-based SC approaches (e.g., in Raft [87] and Paxos [110]), all requests are serialized by a cluster *leader*, in order to provide for a consistent data store view across all cluster *followers*. Thus, with SC, a large distributed system consisting of multiple controller replicas, is effectively constrained into a monolithic system where each data store modification incurs a minimum of two message rounds and a linear message complexity (in the case of a stable leader) in order to synchronize the controller views [3, 181, 182].

In contrast to the correctness benefits of the serialization of state-updates, Raft and Paxos posses the disadvantage of an added overhead in the expected response time and lower availability [3] compared to using the EC primitives. Namely, a single data store update initiated at a follower replica requires a round trip to the leader for confirmation; as well as reaching consensus among the majority of replicas [181]. This leads to an added blocking period and an overhead in confirmation of transactions. Furthermore, quorum-based consensus algorithms can tolerate a maximum of $F = \lceil C/2 - 1 \rceil$ failures in a cluster of C controllers. This limitation relates to the requirement of ensuring data consistency in the case of network partitions, an invariant feasible only when a majority of nodes are involved in confirming the transactions [68, 149]. In the best case, the cluster operates at the speed of the leader, and in the worst case, at the speed of the slowest follower [3].

Eventual Consistency (EC): In EC model [113, 150, 183], state transitions may be delayed and out out-of-order for an arbitrary period of time. Message updates are advertised in a single round and with linear message complexity. From SDN controller perspective, each controller instance in EC is able to autonomously service the client requests. The updates to the internal data store are thus non-blocking and are executed without incurring an additional delay in SDN application's processing time [181]. All reads and writes are performed locally at the processing speed of the local replica. Hence, applications written on top of the EC primitives proceed their operation without a penalty of confirmation time. The state-updates in EC are propagated in the background. In ONOS, for example, state distribution across the EC replicas occurs using an update-push distribution process and the anti-entropy, where replicas continuously compare their local state and eventually converge the deltas. Furthermore, updates to the states may be marked with local timestamps, hence allowing for global ordering of updates. EC favors the performance at the expense of consistency, potentially leading to correctness issues if the applications rely on the *non-staleness* of the local state for their correct operation [113]. Namely, the missing state serialization can result in write conflicts and inefficient decision-making [113].

Adaptive Consistency (AC): AC, similar to EC, realizes the state synchronization as a non-blocking task. However, after exceeding a configurable number of maximum concurrent per-replica state-modifications, an AC system blocks further updates until all replicas have synchronized to a common state [10]. If the system detects that the *staleness* constraints of an SDN application may be violated by a concurrent state-modification, it blocks the future state modifications until the state consistency across all replicas is re-established. Additionally, AC autonomously adapts the Consistency

Level (CL) metric of the system. This adaptation advocates an asynchronous state synchronization at a dynamically decided frequency across the controller cluster. Hence, the maximum number of allowed concurrently executed per-replica transactions varies based on the current SDN application performance observed at runtime. The adaptation mechanism thus optimizes the trade-off between correctness and scalability of controller's decision-making logic.

5.2 Related Work - On Impact of Applied Consistency Model on SDN Performance

Levin et al. [113] evaluate the impact of an inconsistent global network view on controller-based load balancer's performance assuming flexible frequencies of synchronization periods. Their results suggest that an inconsistency in the SDN control state view across multiple controller instances significantly degrades the performance of the SDN applications agnostic to the underlying state distribution. In contrast to our work, the authors generalize the synchronization procedure to a periodic task with flexible periods. In the case of SC, we consider a continuous synchronization model where on-the-wire transactions must be serialized in order to ensure a total ordering of decisions. In the EC case, we assume non-periodic state synchronization, as this provides for a more realistic and better performance and lower penalty of state staleness, especially for the case of higher request arrival rates.

In [33], the authors compare an adaptive approach to the state synchronization between the controllers with an approach of using the non-adaptive controllers that synchronize state with a constant synchronization period. The authors deploy an adaptation module to apply one of the preconfigured fixed synchronization intervals, which makes the approach inflexible for frequent network changes (i.e., varying controller request loads and network congestions). In contrast, we define an adaptation function which controls the new update admissions in order to preserve the worst-case staleness bounds. Our work extends the conclusions on the practical applicability of AC by considering constant re-adaptation of the Consistency Levels (CLs).

TACT middleware [198] enforces consistency bounds among the replicas of a distributed system. To bound the level of inconsistency, TACT defines consistency measures, including: i) the *order error*, which limits the number of tentative writes that can be outstanding at any replica; ii) the *numerical error*, which bounds the difference between the value delivered to the client and the most consistent value; iii) and *staleness*, which places a probabilistic real-time bound on the delay of propagating the writes. A well-known arrival rate for the incoming requests is used to estimate the probabilistic staleness bound.

We distinguish ourselves from TACT by introducing an admission control mechanism for serving new state modifications at any randomly selected serving instance. Thus, we proactively place deterministic bounds on the maximum number of allowed isolated local state-updates.

A corner case of the exceeded limit values for isolated Positive-Negative Counter (PN-Counter) state-updates is related to the issue of conflicting over-reservations. Balegas et al. [35, 36] solve

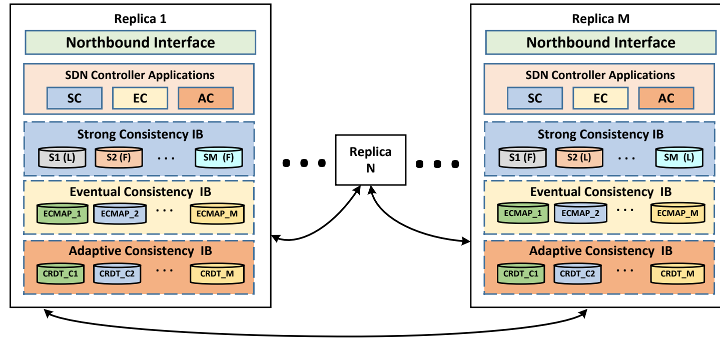


Figure 5.1: The presented internal controller model comprising data stores with varying degrees of state consistency. The controller designer is allowed to select the appropriate data quality based on their application correctness and throughput requirements.

this issue by implementing an escrow-based bounded counter Conflict-Free Replicated Data Types (CRDT), so to guarantee that a value of a counter never exceeds some limit value.

We next provide the insights into the realization of our AC-enabled SDN controller.

5.3 Adaptive Consistency: Relaxing the Consistency Requirement

5.3.1 Design Overview

To guarantee a successful synchronization of the committed state-updates across all replicas, we assume a system with enabled partial synchrony and an *eventual synchronous* communication model. Thus, to make progress in SC and AC (during blocking period) systems, a guaranteed eventual delivery of each state-update to all healthy replicas is assumed. We assume a *fair* and *robust* control channel, where given a non-partitioned network, messages sent infinitely often are delivered infinitely often [150].

Fig. 5.1 presents our envisioned controller design for tolerating Fail-Stop failures. Each controller executes a number of SDN applications (i.e., path-finding, load-balancing), and each controller instance executes a copy of each application (ref. Fig. 5.1). SDN applications base their decisions on the current content of either one or more in-memory data stores which leverage different consistency models.

The controller data store may be partitioned into a number of *data-shards*. For the SC model, we analyze a single default data-shard replicated across each replica, and thus a single Raft session responsible for distribution of all state-updates. EC maps, on the other hand, are data structures whose synchronization is enforced in the background using gossiping / broadcast dissemination. In ONOS [39], for example, EC maps are replicated to all controller instances that are members of a common cluster. Finally, the replication of the AC data structures presented henceforth is handled on a per data-state instance basis, so to allow for granular guarantees on the minimum synchronization interval.

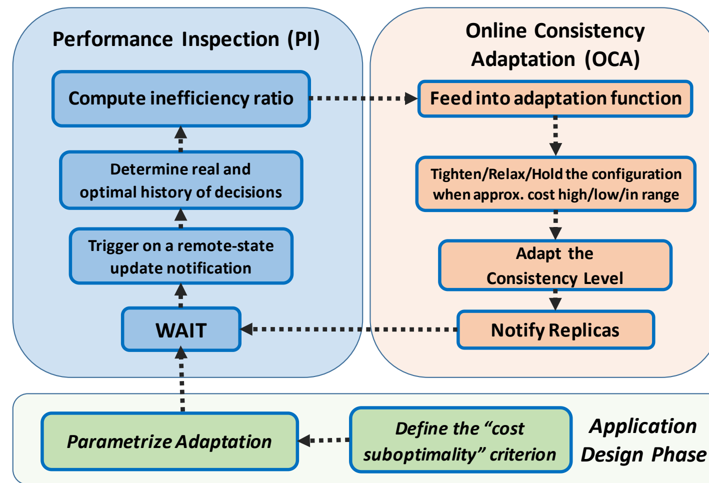


Figure 5.2: High-level architecture of the AC framework. The Performance Inspection Block (PI) block is responsible for the inspection of the negative effect of the state synchronization on the quality of decision-making and the generation of a corresponding inefficiency ratio. The Online Consistency Adaptation Block (OCA) block takes this variable as an input for the adaptation of the CL currently applied for the monitored state.

5.3.2 Relaxation of the Consistency Requirement

The AC framework allows for C.R.U.D. operations on the granular state instances (such as counters, registers, maps etc.). The operations are *eventually* synchronized between the controller replicas. However, in contrast to EC, that allows for enqueueing of an unbounded number of buffered unconfirmed operations, the maximum number of enqueued manipulations in an AC framework is limited by the size of an *update distribution queue* and a *timeout-based automated distribution* of the enqueued updates. The maximum size of the state-update distribution queue and the maximum timeout duration are governed by the currently applied CL. The maximum distribution queue size and the timeout are maintained at the granularity of an observed data store state. Given an application "inefficiency" metric and the optimization target, the AC framework decides on the optimal CL to be applied for upcoming operations.

The high-level process flow of the AC framework is depicted in Fig. 5.2.

Application Design Phase: The application designer writes an SDN application built atop of the AC framework and parametrizes the adaptation function. The developer must furthermore present an *inefficiency metric* related to his application logic (e.g., the optimality of routing decisions [10]), as well as parametrize the adaptation thresholds / efficiency targets. The specification parameters of the adaptation target vary with the choice of the adaptation function. We discuss the threshold- and Proportional, Integral, Derivative Control (PID)-based functions in Section 5.3.6.

Performance Inspection Block (PI) Phase: At time t_L, the PI block triggers on an eventually delivered state-update U, initiated by a remote SDN controller replica C_R. In the PI block, each controller replica C_L locally decides its own view of the real history of state-updates, by ordering the updates based on the time-stamp of updates. Let t_U denote the time when update U was initiated at C_R. Then, after deciding the global order of updates, each replica C_L evaluates the effect of being

late-notified of the update U. To do so, it compares two histories: i) the set of results associated with the actual actions taken during the period $[t_U, t_L]$; ii) The set of results associated with the actions that would have been taken during the period $[t_U, t_L]$ *if* the update U had been serialized and known to C_L at t_U. Thus, two sets of results are stored, the set of real (i.e., suboptimal) decisions, and the set of ideal (i.e., optimal) decisions.

Online Consistency Adaptation Block (OCA) Phase: An *inefficiency* metric (i.e., an approximation factor), estimates the ratio between the suboptimal and optimal decision, and thus the cost of eventual state-update delivery. The latest measured inefficiency is fed into the OCA. In order to decide on the best-fitting CL for the observed state, the OCA considers the latest- and, optionally, the *history* of previous inefficiency reports. The OCA then disseminates this decision to all cluster replicas. The overhead of the OCA block is hence centralized at a single controller that collects the remote replica's inefficiency metrics and decides on the CL. PI and OCA blocks are pipelined for a particular state, but are parallelized for updates on different state instances, thus enabling scalable consistency adaptation.

In the following sections, we present the mechanisms behind our prototype AC framework realization, comprising: i) a CRDT-based in-memory data store; ii) a generalized SDN-based Application Load Balancer (SDN-LB); iii) the corresponding PI block; iv) the OCA block comprising the threshold- and PID-based adaptation functions; and v) the mechanism for cluster-wide state synchronization.

5.3.3 On Use of CRDTs for State Modeling

Conflict-Free Replicated Data Types (CRDT) [174] are a novel approach to handling conflict-free distributed updates on a set of eventually consistent data structures. With CRDTs, the isolated views on a single CRDT instance in remote replicas eventually converge to the same value, independent of the order of updates. Thus, CRDTs preserve the correctness invariant, even in the case of an increased network latency and packet loss. Updates to a CRDT monotonically advance according to a partial order, subsequently converging towards the least upper bound of the most recent value.

An example of a replicated counter datatype is a PN-Counter whose increment and decrement manipulations commute. Our take on a PN-Counter realization is presented in Alg. 1. Our AC implementation leverages CRDT registers and set structures as well. For brevity, however, we present only the PN-Counter here, and refer to [174] for an overview of other data-types.

In AC, individual state-updates are synchronized across the controller replicas, and are stored in a log-tree, together with their initiation time-stamp, for the purpose of later reference in the PI block. The accepted updates to a CRDT-modeled state are synchronized across the controller replicas, while rejections result in data store update failures and subsequent notifications to the requesting applications. The admission control for new updates is based on the properties of the queue distribution (i.e., the maximum queue size), governed by the currently applied CL associated with the target CRDT (ref. Section 5.3.7).

Alg. 1 presents our PN-Counter realization. The respective notation used in Alg. 1 and Alg. 2 is explained in Table 5.1. Upon a new client request to modify a particular data store object (realized as a counter CRDT instance), the local controller executes the admission control (Lines 3-9). If the update

is accepted, the controller MERGEs the update with its local CRDT (Lines 13-17). It then enqueues the update for a cluster-wide distribution (ref. Section 5.3.7). On receiving the update initiated by a remote controller, the local controller executes the MERGE function (Lines 22-28). Each CRDT additionally implements the QUERY function, allowing to read its current state.

Parameter	Description
C_R	Remote controller replica
C_L	Local controller replica
B_j	Client requesting a CRDT state-update
Ctr_k	PN-Counter targeted for update
$S_{Ctr}^{C_L}$	Set of PN-Counters stored in C_L's AC data store
U_{Ctr_k}	Update request for state Ctr_k
$B[B_j, Ctr_k]$	Update-log for client B_j and state Ctr_k
$U_{Ctr_k}^{remote}$	Reported remote update request for state Ctr_k
$U_{Ctr_k}^{local}$	Local update request for state Ctr_k
$S_{Ctr_k^U}$	Set of previously logged updates for state Ctr_k
$U(T)$	Timestamp of the state-update U at C_R
$U(R)$	Client request that resulted in the update U
CL_{Ctr_k}	Currently applied CL for counter Ctr_k
$S_{\phi}^{Ctr_k}$	Set of previously stored inefficiency reports
T_U	Upper adaptation trigger for the threshold metric
T_L	Lower adaptation trigger for the threshold metric
W	Window size of considered inefficiency observations
T_O	Target oscillation value
I_g, P_g, D_g	Integral, proportional and differential gains
$CL_{QS}^{Ctr_k}$	Max. distribution queue size for the applied CL
$CL_{TO}^{Ctr_k}$	Distribution timeout for the applied CL
$Q_{Ctr_k}^{E}$	Distribution queue for the unacknowledged state-updates committed at the local replica
QC_{Ctr_k}	List of local state-updates acknowledged by all remote replicas
C	The set of remote controller replicas

Table 5.1: Notation used in Alg. 1, Alg. 2, Alg. 3, Alg. 4 and Alg. 5.

5.3.4 Performance Inspection Block

The adaptation of the CLs for a particular state is based on a provided application *inefficiency* metric. The inefficiency metric is the approximation ratio between the series of *observed* and *optimal* results of an SDN controller application's decisions. The *optimal* result comprises the decisions the application would have made if each update in the system had been serialized. The *observed* result is the one the local replica has achieved based on its local state, without consideration of the status of other replicas. For a replica to compute the optimal result, the knowledge about the content and timing of the eventually delivered updates must be available. The timing characteristics are necessary for the total ordering of the observed and eventually delivered state-updates.

The generalized calculation of the inefficiency metric is depicted in Alg. 2. In Lines 6-8 the PI block identifies the previously executed operations on the observed state, in the period prior to initiation of the *remote* update $U_{Ctr_k^{remote}}$ at the remote replica C_R. Thus, Line 8 yields an array of consistent entries which correspond to a part of the serialized true history of updates $S_{U_{cnst}}$. Lines 10-12 identify the set of client requests that have resulted in *potentially* suboptimal decisions, made in the past by the local replica C_L, without the consideration of the eventually delivered remote state-update. Lines 14-16 derive the application-specific optimal decisions, given the identified optimal history $S_{U_{cnst}}$, the serialized remote update $U_{Ctr_k}^{remote}$ and a set of client requests $R_{U_{incnst}}$, previously served in a suboptimal

Algorithm 1 Distributed CRDT PN-Counter.

1: **upon event** *client-update*$< B_j, U_{Ctr_k} >$ **do**
2: **if** $Ctr_k \in S_{Ctr}^{C_L}$ **then**
3: $success := evalAddToDistributionQueue(U_{Ctr_k})$
4: **if** $success$ == *True* **then**
5: $B[B_j, Ctr_k] \leftarrow B[B_j, Ctr_k] \cup U_{Ctr_k}$
6: $merge(U_{Ctr_k})$
7: $notify(B_j,$ *update-success* $< U_{Ctr_k} >)$
8: **else**
9: $notify(B_j,$ *update-failed* $< U_{Ctr_k} >)$
10: **else**
11: $notify(B_j,$ *update-failed* $< U_{Ctr_k} >)$
12:
13: **function** MERGE(U_{Ctr_k})
14: **if** $U_{Ctr_k}.operation == DECREMENT$ **then**
15: $Decr[Ctr_k] \leftarrow Decr[Ctr_k] \cup U_{Ctr_k}.amount$
16: **else if** $U_{Ctr_k}.operation == INCREMENT$ **then**
17: $Incr[Ctr_k] \leftarrow Incr[Ctr_k] \cup U_{Ctr_k}.amount$
18:
19: **function** QUERY(Ctr_k)
20: *return*($\sum_j Incr[Ctr_k]_j - \sum_j Decr[Ctr_k]_j$)
21:
22: **upon event** *remote-update*$< C_R, < B_r, U_{Ctr_k} >>$ **do**
23: **if** $Ctr_k \notin S_{Ctr}^{C_L}$ **then**
24: $notify(C_R,$ *update-failed* $< U_{Ctr_k} >)$
25: **else if** $Ctr_k \in S_{Ctr}^{C_L}$ **then**
26: $B[B_r, Ctr_k] \leftarrow B[B_r, Ctr_k] \cup U_{Ctr_k}$
27: $merge(U_{Ctr_k})$
28: $notify(C_R,$ *update-success* $< U_{Ctr_k} >)$

manner. The method `CompInefficiency()` in Line 20 takes as an argument the consistent (optimal) history of decisions $S_{U_{cnst}}$, and the actual, potentially suboptimal, history of decisions $S_{U_{incnst}}$. It then returns the inefficiency (i.e., the approximation ratio) ϕ given the two series of decisions.

Let σ_u^R and σ_o^R denote the cost of suboptimal and optimal decisions for a request R, respectively. The binary value of X_{subopt}^R denotes an *inefficient* result, induced by the staleness (caused by synchronization delay):

$$X_{subopt}^R = \begin{cases} 1 & \text{when } \sigma_u^R > \sigma_o^R \\ 0 & \text{when } \sigma_u^R \leq \sigma_o^R \end{cases}$$

`CompInefficiency()` and `AppLogic()` functions are application-specific implementations. In the next section, we present an exemplary `CompInefficiency()` realization of an SDN-LB [138]. Its `AppLogic()` realization is assumed to optimally assign each incoming client request to the replicated server instances based on its *current local* view of the server resource utilizations. We evaluate the algorithm in Section 5.4.

Algorithm 2 Inefficiency calculation for a distributed CRDT.

1: **procedure** HANDLE NEW COUNTER UPDATE
2: **upon event** *update* $< C_R, U_{Ctr_k^{remote}} >$ **do**
3: $S_{U_{cnst}} := \emptyset$
4: $S_{U_{incnst}} := \emptyset$
5:
6: **for all** $U_{Ctr_k}^{local} \in S_{Ctr_k^U}$ **do**
7: **if** $U_{Ctr_k}^{local}(T) < U_{Ctr_k}^{remote}(T)$ **then**
8: $S_{U_{cnst}} \leftarrow S_{U_{cnst}} \cup U_{Ctr_k}^{local}$
9:
10: **for all** $U_{Ctr_k}^{local} \in S_{Ctr_k^U}$ **do**
11: **if** $U_{Ctr_k}^{local}(T) >= U_{Ctr_k}^{remote}(T)$ **then**
12: $R_{U_{incnst}} \leftarrow R_{U_{incnst}} \cup U_{Ctr_k}^{local}(R)$
13:
14: **for all** $R_U \in R_{U_{incnst}}$ **do**
15: $U_{Ctr_k}^{localOpt} := AppLogic(R_U, S_{U_{cnst}})$
16: $S_{U_{cnst}} \leftarrow S_{U_{cnst}} \cup U_{Ctr_k^{localOpt}}$
17:
18: $S_{U_{cnst}} \leftarrow S_{U_{cnst}} \cup U_{Ctr_k}^{remote}$
19: $S_{U_{incnst}} \leftarrow S_{Ctr_k^U} \cup U_{Ctr_k}^{remote}$
20: $\phi = CompInefficiency\ (S_{U_{incnst}}, S_{U_{cnst}})$
21:
22: $reportIneff(\phi)$

5.3.5 An Exemplary Inefficiency Metric for an Online SDN-LB

For a set of defined data store states S and a state-update $U(t-n)$, timestamped at time $t-n$, let $T(t-n)$ be the matrix of observations of the states encompassing the period $[t-n\ ..\ t]$. Then, $T(t-n)$ is an $|S| \times n$ matrix.

Let $S(i)$ be the i-th vector of the observed state values at time instant $t-n+i$, so that:

$$\mathrm{S}(i) = (s_1(i), s_2(i)..s_m(i)) : S(i) \in T(t-n) \text{ s.t. } N_{res}(i) = |S(i)|$$

First, let $S_{U_{incnst}}$ contain the suboptimal (real) history of state-updates. Let $S_{U_{cnst}}$ accordingly hold the computed ideal (optimal) history of state-updates (computed as per Lines 18-19 of Alg. 2). Then, for each vector (time-point) i of observed state values $S_{U_{incnst}}$ and $S_{U_{cnst}}$ we can compute the costs of optimal and suboptimal decisions at time i, $\sigma_o^{R_i}$ and $\sigma_u^{R_i}$ respectively, using the standard deviation:

$$\sigma_o^{R_i} = \sqrt{\frac{1}{N_{res}(i)} \sum_{j=1}^{N_{res}(i)} (s_j(i) - \mu_{S_{cnst}^{R_i}})^2} \text{ where } s_j(i) \in S_{U_{cnst}}(i)$$

$$\sigma_u^{R_i} = \sqrt{\frac{1}{N_{res}(i)} \sum_{j=1}^{N_{res}(i)} (s_j(i) - \mu_{S_{incnst}^{R_i}})^2} \text{ where } s_j(i) \in S_{U_{incnst}}(i)$$

where

$$\mu_{S^{R_i}} = \frac{\sum_{j=1}^{N_{res}(i)} s(j)}{N_{res}(i)} \tag{5.1}$$

represents the mean utilization of resources at time-offset i, and R_i represents the client request at time-offset i. After computing the costs of optimal- and suboptimal decisions, the average inefficiency Φ_T for the observation interval $T(t-n)$ is defined as:

$$\Phi_T = \frac{\sum_{i=0}^{\|T\|} \sigma_u^{R_i}}{\sum_{i=0}^{\|T\|} \sigma_o^{R_i}}$$

5.3.6 Online Consistency Adaptation Block

The OCA block is responsible for the collection of computed inefficiency values and their online evaluation. The output of the OCA block is the adapted CL for the observed state instance. The computed inefficiency value ϕ is input into the OCA block and the adaptation function `reportIneff()` is called, as depicted in Fig. 5.2 and Alg. 2, respectively. We next present two methodologies for adapting the CL, given a historical set of inefficiency reports:

- **Threshold-based CL adaptation**: If the observed mean inefficiency over a window of inefficiency observations of size W is below, above or in between the lower and upper thresholds, the adaptation function decides to raise, lower or keep the currently applied CL, respectively. Threshold-based CL adaptation is specified in Alg. 3.
- **PID-based CL adaptation**: In addition to the *integral* part, the PID-based feedback compensation also considers *proportional* and *differential* parts of the recent inefficiency reports. Each part can be assigned a corresponding weight, thus allowing to favor either a fast adaptation response or long-term adaptation accuracy. Alg. 4 describes the procedure.

Algorithm 3 Threshold-based CL Adaptation.

1: **procedure** HANDLE NEW INEFFICIENCY REPORT
2: **function** REPORTINEFF(ϕ)
3: $S_\phi^{Ctr_k} \leftarrow S_\phi^{Ctr_k} \cup \phi$
4: $S_{RIv} := S_\phi^{Ctr_k}[|S_\phi^{Ctr_k}| - W : |S_\phi^{Ctr_k}|]$
5: $\mu_{S_{RIv}} := \frac{\sum_{i=0}^{|S_{RIv}|} S_{RIv}(i)}{|S_{RIv}|}$
6:
7: **if** $\mu_{S_{RIv}} \geq T_U$ **then**
8: *raiseCL* (CL_{Ctr_k})
9: **else if** $\mu_{S_{RIv}} \leq T_L$ **then**
10: *lowerCL* (CL_{Ctr_k})

Algorithm 4 PID-based CL Adaptation.

```
1:  procedure HANDLE NEW INEFFICIENCY REPORT
2:      function REPORTINEFF(ϕ)
3:          S_ϕ^{Ctr_k} ← S_ϕ^{Ctr_k} ∪ ϕ
4:          P_term := P_g · (T_O − S_ϕ^{Ctr_k}[|S_ϕ^{Ctr_k}|])
5:          I_term := I_g · Σ_{i=|S_ϕ^{Ctr_k}|−W}^{|S_ϕ^{Ctr_k}|} (S_ϕ^{Ctr_k}_i − T_O)
6:          D_term := D_g · [(S_ϕ^{Ctr_k}[|S_Φ^{Ctr_k}|] − T_O) − (S_ϕ^{Ctr_k}[|S_ϕ^{Ctr_k}| − 1] − T_O)]
7:
8:          T := P_term + I_term + D_term
9:          if T > T_O then
10:             raiseCL (CL_{Ctr_k})
11:         else if T < T_O then
12:             lowerCL (CL_{Ctr_k})
```

5.3.7 State Synchronization Strategies

To limit the amount of unseen updates for a particular state on a diverged controller replica, AC ensures that a reliable distribution of a bounded set of updates has occurred before a new data store transaction for the target state is allowed in the system. Each time a client requests a new state-update, we evaluate the number of previously submitted unconfirmed state-updates on the local replica. If this number is above the maximum queue size governed by the currently applied CL, the transaction is rejected. Otherwise, the state-update is enqueued in a per-state First-In, First-Out (FIFO) queue. We distinguish two abstractions of state-update distribution abstractions, with unique trade-offs in terms of response time and the generated control plane load. Both abstractions ensure limited staleness by bounding the maximum number of enqueued isolated updates per replica:

- **Fast-Mode State Distribution** The first procedure of Alg. 5 realizes this distribution abstraction. If the actual occupancy of the state distribution queue is below the CL-governed threshold, the state-update is *admitted* for processing (Lines 3-6), otherwise it is dropped (Lines 7-8). Admitted updates are prepared for the distribution to other cluster members. The unconfirmed updates in the system are first enqueued in the distribution queue (Line 4). The distribution procedure then serializes outstanding unconfirmed updates and distributes these to remote replicas (Line 5). The sender replica then waits on the asynchronous confirmations for the individual updates. After all *active* replicas have acknowledged the state-update(s), the sender removes the *acknowledged* updates from the distribution queue.

- **Batched-Mode State Distribution** The second procedure of Alg. 5 realizes this distribution abstraction. The transmission of a series of unconfirmed updates on each new update has the advantage of the lowered response time and reliability in the case where some of the previously sent out packets are lost. Nevertheless, generation of a new frame for each new state-update may cause unnecessary load if the response time is not the optimization criterion. For such scenarios, we have realized a batching queue that collects a number of state-updates (Line 4), up to the maximum amount defined by the applied CL for the particular state, and distributes these in a batch to the peer replicas (Line 7). For infrequently updated state-instances, we introduce an

asynchronous timer that triggers the state-update distribution whenever a non-empty queue is not distributed for the duration of time specified by the applied CL (Lines 14-17).

Algorithm 5 Fast-Mode and Batched-Mode distribution of state-updates.

1: **procedure** FAST-MODE DISTRIBUTION
2: **function** EVALADDTODISTRIBUTIONQUEUE($U_{Ctr_k}^{local}$)
3: **if** $occupied(Q_{Ctr_k}^E) < CL_{QS}^{Ctr_k}$ **then**
4: $enqueue(Q_{Ctr_k}^E, U_{Ctr_k}^{local})$
5: $distribute(C, Q_{Ctr_k}^E)$
6: *return* True
7: **else if** $occupied(Q_{Ctr_k}^E) \geq CL_{QS}^{Ctr_k}$ **then**
8: *return* False
9:
1: **procedure** BATCHED-MODE DISTRIBUTION
2: **function** EVALADDTODISTRIBUTIONQUEUE($U_{Ctr_k}^{local}$)
3: **if** $occupied(Q_{Ctr_k}^E) < CL_{QS}^{Ctr_k}$ **then**
4: $enqueue(Q_{Ctr_k}^E, U_{Ctr_k}^{local})$
5: **if** $occupied(Q_{Ctr_k}^E) == CL_{QS}^{Ctr_k}$ **then**
6: $T_{Ctr_k} == null$
7: $distribute(C, Q_{Ctr_k}^E)$
8: **if** $T_{Ctr_k} == null$ **then**
9: $T_{Ctr_k} := init\text{-}timer(CL_{TO}^{Ctr_k})$
10: *return* True
11: **else if** $occupied(Q_{Ctr_k}^E) \geq CL_{QS}^{Ctr_k}$ **then**
12: *return* False
13:
14: **upon event** $expired < T_{Ctr_k} >$ **do**
15: $T_{Ctr_k} == null$
16: **if** $occupied(Q_{Ctr_k}^E) > 0$ **then**
17: $distribute(C, Q_{Ctr_k}^E)$
1: **procedure** ON ACKNOWLEDGMENT OF DISTRIBUTION (FAST- AND BATCHED-MODE)
2: **upon event** $acknowledged < QC_{Ctr_k} >$ **do**
3: **for all** $U_{Ctr_k}^{local} \in QC_{Ctr_k}$ **do**
4: $Q_{Ctr_k}^E$.remove($U_{Ctr_k}^{local}$)

5.4 Evaluation and Results: Comparison of AC, EC and SC in Distributed SDN Control Plane

SDN Load Balancing: To present the trade-offs in deploying either SC, EC and AC in a multi-controller testbed, we have implemented and evaluated a distributed load-balancer application (SDN-LB) as a component of a modified ODL [123] distribution. The SDN-LB allows for the embedding of isolated independent services via a Yet Another Next Generation (YANG)-modeled Representational State Transfer (REST) interface, characterized by the type and cost (i.e., comprising a capacity requirement). A data-plane SDN-LB has been investigated in the past in the context of the link-load distribution scenarios [113, 169]. However, we generalize the goals of the SDN-LB to support allocation of any type of resources (i.e., bandwidth / CPU / memory) on the selected optimal service node, given a subset

of feasible candidate nodes and their current utilizations and capacities as inputs. The algorithm then decides to assign the service request on the node deemed as optimal w.r.t. total balance of resource utilization. We adapt the algorithm defined in [138] to facilitate immediate scheduling, i.e., an online resource mapping process.

SDN Path Computation: We evaluate the inefficiency of path finding computation in an AC setup, on example of a cluster-aware path-finding SDN application. The projected inefficiency is a function of number of controller replicas, the active CL, client traffic model and the network topology size. The application utilizes a bandwidth-constrained Dijkstra implementation to identify and reserve paths for uniformly selected source and destination pairs in a variable-size grid network. Due to the space constraints, however, we do not include this contribution here, but instead refer the interested reader to [10].

We model the state of the current reservations and the available resources as in-memory state instances in our data store realizations. SDN-LB decisions are made based on the current value of these states. Upon each successful embedding, the current node utilization is updated to include the cost of the latest request. The controller is then in charge of disseminating the local reservation update using the update-distribution and commit mechanism implemented by the underlying data store.

In the case of SC, every update in the distributed data store is serialized by the Raft leader. In EC and AC, each new resource reservation necessitates an update to the respective CRDT PN-Counter (ref. Alg. 1). Combined with the commutative increment / decrement operations, the PN-Counter ensures eventual convergence in both models and thus represents a good data structure fit for resource tracking realization. The updates to the PN-Counter in AC are queued and distributed across the cluster based on the CL-timer and maximum queue-distribution thresholds, governed by the currently applied CL. The adaptation function (ref. Section 5.3.6) adapts the CL, and thus manipulates the worst-case time required to synchronize the counters. Thus, the adaptation affects the quality of the embedding when using AC. The inefficiency metric provided as an input for the consistency adaptation ϕ maps to the approximation ratio introduced in Section 5.3.5. In EC, state-updates are queued in a state-specific FIFO distribution queue and are distributed without waiting.

Data Store Implementations: To empirically compare the effects of deploying the different consistency models, we have implemented and integrated in ODL the three data stores:

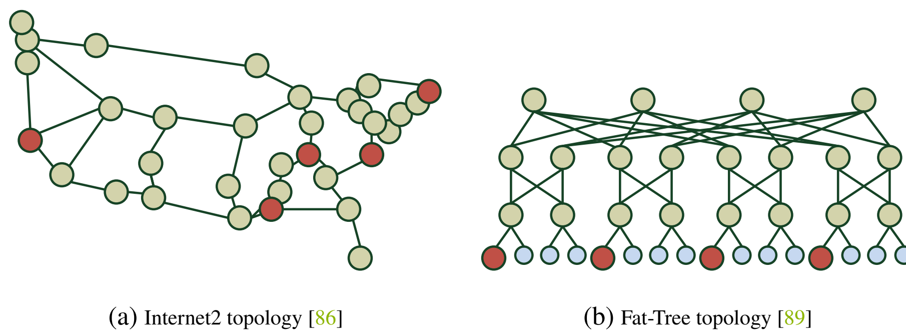
- SC: The SC data store is realized using the unmodified Raft [87] implementation atop of Java and Akka.io[1] concurrency framework included in the ODL's Boron-SR4 distribution. We have modeled the data-models required by the generic SDN-LB using YANG[2]. We then synthesized these models into REST APIs using ODL's YANG Tools[3] compiler.
- AC: The AC implementation is based on the algorithms presented in Section 5.3.2. The framework is implemented as a set of Java bundles, and has been integrated in ODL's OSGi

[1] Akka Clustering and Remoting - https://akka.io/
[2] YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) - https://tools.ietf.org/html/rfc6020
[3] YANG Tools - https://wiki.opendaylight.org/view/YANG_Tools:Main

environment as an in-memory data store in parallel to the SC. In AC, as above, we expose the data-model for the SDN-LB application using the YANG and REST APIs. Additionally, the CL adaptation as well as the distribution of the CRDT state-updates (Section 5.3.7) require a new protocol definition. We have used Google Protobuf[4] to describe the data structures and serialize the on-the-wire transmissions.

- EC: In the AC realization, the update-distribution queue thresholds are derived from the specification of the currently applied CLs. In EC, however, the CLs hold no relevance for the state distribution, hence the maximum queue sizes of the distributed state-updates are unbounded and can grow infinitely for high request arrival rates.

On topology and parameter selection: We base our evaluation of the consistency models using an in-band OF control plane and an emulated forwarding plane, consisting of a number of interconnected OVS instances isolated in individual Docker containers. We have emulated the Internet2 as well as a standard Fat-Tree data-center topology, controlled by a 5- and 4-controller cluster, respectively. To reflect the delays incurred by the length of the optical links in the geographically scattered Internet2 topology, we assume a travel speed of light of $2 \cdot 10^6 km/s$ in the optical fiber links. We derive the link distances and hence the propagation delays from the publicly available geographical Internet2 data [225]. The links of the Fat-Tree topology were modeled to incur a variable propagation and processing delays averaging $1ms$. In the Internet Service Provider (ISP) topology we leveraged a controller placement that targets the maximized robustness against the controller failures and a minimization of the probability of occurrence of switch partitions, as per the optimal placement in [86, 112]. The resulting controller placement is depicted in Fig. 5.3a. SDN controller replicas in the data-center topology are assumed to run on the leaf-nodes, deployed as VMs [89] (Fig. 5.3b).



(a) Internet2 topology [86]　　(b) Fat-Tree topology [89]

Figure 5.3: Exemplary network topologies and controller placements used in the SC, AC and EC evaluation. Elements in green and blue represent the forwarding and compute devices, respectively. Red elements are the ODL instances placed as per [86, 89].

We model the arrival rates of the incoming service embedding requests using a N.E.D. [96]. To emphasize the effects of the EC on the quality of decision-making in the SDN-LB, we distribute the total request load non-uniformly across the replicas. The arrival rates and the per-replica load weights are included in Table 5.2. The Docker- and OVS-based topology emulator, as well as the 5

[4]Google Protocol Buffers - https://developers.google.com/protocol-buffers/

Parameter	Model	Value	Comment
Number of Replicas	SC, AC, EC	$[4^{\cdot}, 5^{+}]$	Internet2^{+} and Fat-Tree$^{\cdot}$
CLs (Granularity)	AC	[1..10]	Ref. Alg. 3 and 4
$CL_{QS}^{Ctr_k}$	AC	[3..15]	Ref. Alg. 5
$CL_{TO}^{Ctr_k}$	AC	[100..1000]	Ref. Alg 5
Initially applied CL	AC	3	N/A
P_g	AC	0.2	Ref. Alg. 4
I_g	AC	0.2	Ref. Alg. 4
D_g	AC	0.1	Ref. Alg. 4
T_O	AC	2	Ref. Alg. 4
W	AC	5	Ref. Alg. 3 & 4
T_L	AC	1.5	Ref. Alg. 3
T_U	AC	3.5	Ref. Alg. 3
$SDN - LB_{\#C}$	SC, AC, EC	2	SDN-LB - No. service types
$SDN - LB_{\#S}$	SC, AC, EC	2	SDN-LB - No. servers
$SDN - LB_{C_{Cost}}$	SC, AC, EC	[500..600]	SDN-LB - Service cost
$C_{Weights}$	SC, AC, EC	$[1, 1, 2, 1, 5]^{+}$ $[1, 2, 2, 5]^{\cdot}$	Req. load for Internet2^{+} and Fat-Tree$^{\cdot}$ topologies

Table 5.2: Parametrizations used in experimental study of different consistency models.

controller replicas, were deployed on a single commodity Personal Computer (PC) equipped with a recent multi-core AMD CPU and 32 GB of Random Access Memory (RAM).

5.4.1 Response Time Impact of Consistency Model Selection

We define the control plane response time as the time duration required to accept, distribute and confirm a single new state-update in the underlying distributed controller data store. Fig. 5.4 depicts the box-plots for the measured commit times in the case of SC and AC models. *AC (local)* showcases the time required to apply an update to the local replica and return a corresponding acknowledgment at the requesting application and is in this regards equivalent to a single-replica and EC deployment model. The *AC (W=3)* case corresponds to the time duration needed to converge the state-update request at 3 of 5 replicas. Thus, in the case of failure or partitioning of a single replica, the remaining replicas can converge on the latest state value, providing for equivalent failure tolerance capability as the SC model.

The analogue case for the SC replicas is depicted in the two right-most box-plots. An SC cluster of 5 replicas similarly tolerates a maximum of 2 controller failures (due to the majority constraint imposed by the CAP theorem [149]). Compared to the *AC (W=3)* case, the SC model offers the advantage of the serialized data stores updates. This benefit, however, comes at a high cost of minimum, average and worst case response time, especially when state-update requests are received at a follower. An incoming state-update request at the leader leads to the faster commit confirmations, as one less uni-directional packet-transmission delay from the followers to the leader is required.

The worst-case commit time in the *AC (W=5)* case is similar to the optimistic SC case. It results in a commit time increase, due to the geographical separation of the controllers in the Internet2 topology. The *SC (Follower)* case considers the update requests received at one of the followers that require transmission and subsequent serialization at leader, as well as the majority consensus to commit the update, leading to an increased resulting response time.

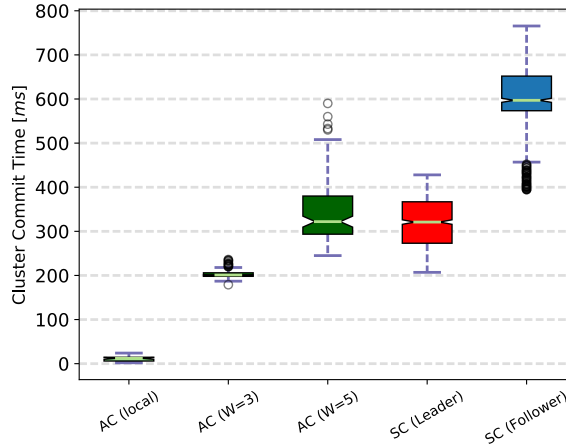
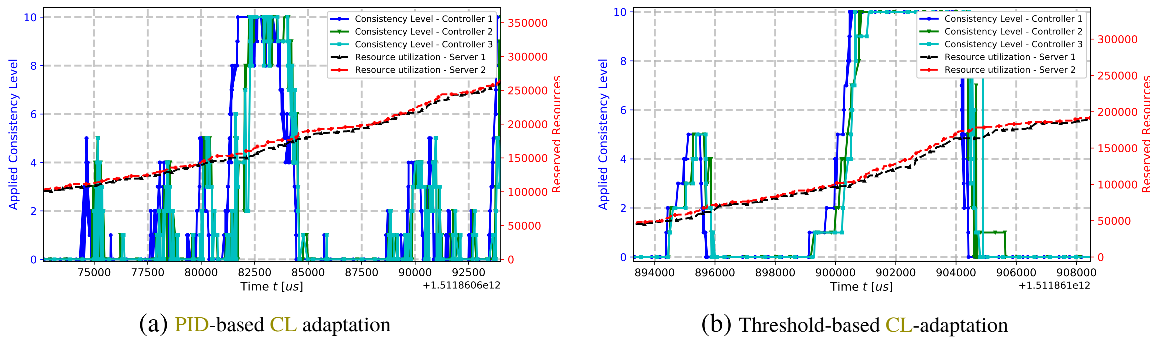


Figure 5.4: The response time of the AC and SC data stores in Internet2. The best-case response time in AC corresponds to the *AC (local)* case (a non-blocking update). Compared to SC configuration with 5 controllers, *AC (W=3)* configuration depicts a better worst-case commit-time performance, while providing for same availability in case of intermediate replica failures. The *AC (W=3)* and *AC (W=5)* blocking periods, however, only occur when the state-update queue is filled and a distribution is necessary to guarantee the *staleness* bound. In contrast, the throughput of the SC cluster suffers specifically in the case where updates are committed on the Raft nodes assigned a follower role, thus confirming results of [181].

5.4.2 Correctness of Application Decision Making

Fig. 5.5 visualizes an exemplary adaptation process in the AC framework. In particular, blue, green and cyan lines depict the CL applied for the SDN-LB-related PN-Counter instances on three different controller replicas. Red and black lines correspond to the actual capacity assignments managed by the SDN-LB instances on the different replicas. The resources are assigned on two different servers providing for the utilizable capacity. In case of a strongly consistent SDN-LB (SC), the black and red lines continuously overlap as the state of reservations is serialized and an optimal placement executes for each incoming service embedding request. Fig. 5.5b highlights the CL adaptation at 905000 us, where the imbalance (i.e., inefficiency) of the SDN-LB leads to an adaptation trigger and a steep decrease of the utilized CL from 10 (most relaxed) to 0 (most strict). The CL hence adaptation modifies the maximum available queue capacity $CL_{QS}^{Ctr_k}$ and worst-case timeout $CL_{TO}^{Ctr_k}$, as per Table 5.2. With the strictness of the CLs, the SDN-LB resource assignment discrepancy decreases, but the overhead of blocking time for new state-updates increases.

Fig. 5.6 depicts the measured inefficiencies in the SDN-LB's assignment for the case of AC- and EC models in Fat-Tree. AC uses the threshold-based adaptation of the CLs, and depicts a faster convergence in the case of the Fast-Mode based AC update distribution, compared to EC. Indeed, the Fast-Mode converges first to the worst-case inefficiency for all depicted request arrival rates. The Batched-Mode shows a lower average inefficiency than the EC model in the case of the request arrival rates of 2 ms. For less frequent 5 ms arrivals, Batched-Mode state-update distribution shows higher mean inefficiency than the EC case. This is due to Batched-Mode collecting and distributing the outstanding state-updates only after a queue-threshold is exceeded or a scheduled CL-governed timer



(a) PID-based CL adaptation

(b) Threshold-based CL-adaptation

Figure 5.5: PID-based (a) and threshold-based (b) adaptation of CLs. The PID-based approach is more volatile compared to a rather resistant threshold-based approach. As per Alg. 3 the threshold-based approach keeps the current CL unmodified, as long as the measured inefficiency stays in a specific range (i.e., between the specified upper and lower thresholds). The PID-based approach, on the other hand, oscillates around the specified target inefficiency value.

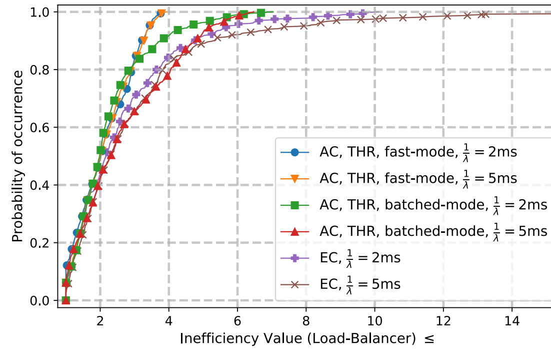


Figure 5.6: ECDFs of reported inefficiencies for various deployed consistency models and request arrival rates in the Fat-Tree topology (ref. Fig. 5.3b). High inefficiency values indicate a more unbalanced performance of the distributed SDN-LB instances. Fast-Mode configurations result in the lowest worst-case inefficiency values. For the Batched-Mode distribution, the system has a similar average inefficiency as the EC. This is related to the delayed distribution of the state-updates, which is initiated only once the distribution queue is fully utilized. In the case of less frequent request arrivals the distribution queue takes the longest to fill up.

has expired. However, the time duration taken to fill up the distribution queue for the Batched-Mode is inversely proportional to the rate of update arrivals. Hence, compared to EC, the staleness caused by slow request arrivals offsets the benefits of bounding the concurrent state-updates.

Fig. 5.7 portrays the comparison of threshold- and PID-based AC consistency adaptation, as well as the EC consistency model in the case of the Internet2. Again, the usage of bounded state-update distribution queues leads to a bounded worst-case inefficiency with both adaptation functions. Interestingly, threshold-based adaptation depicts a lower average- and worst-case performance for the estimated inefficiency, compared to PID model. This behavior is caused by the tendency of the PID-model to relax the state synchronization frequency more often, thus leading to a slightly higher inefficiency, at the benefit of higher transaction throughput.

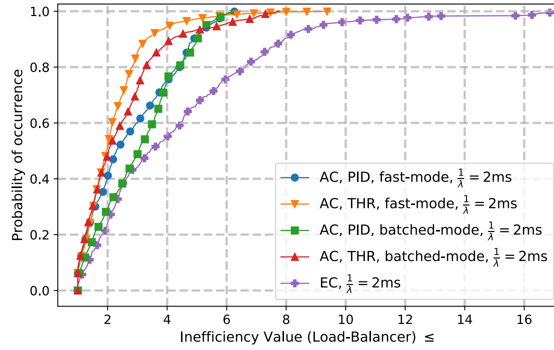


Figure 5.7: ECDFs of the reported inefficiencies for the Internet2 topology. The average- and the worst-case inefficiencies of the EC model are larger than in the case of Fat-Tree. This is due to larger C2C network delays in the geographically spaced Internet2. The AC model deployment does not suffer from this issue because of the limited amount of incurable staleness, guaranteed by the maximum amount of unsynchronized updates. Threshold-based adaptation model shows a better performance than PID-based adaptation.

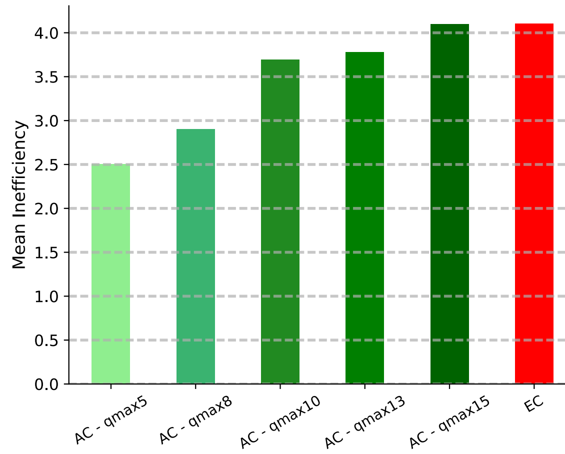


Figure 5.8: The measured mean inefficiency for the different $CL_{QS}^{Ctr_k}$ parametrizations for the most-relaxed CL. The larger the maximum allowed state-update queue size, the larger the sojourn time of the locally admitted state-updates without enforced data store synchronization. Thus, the inefficiency of the system scales with the number of unseen state modifications.

Fig. 5.8 emphasizes the effect of the design of CL configurations on the average-case inefficiency. The experienced worst-case inefficiency scales with the number of allowed isolated state-updates at a single replica. Hence, careful parametrization of the CL mappings to the maximum state-update distribution queue sizes and the timer duration is necessary to ensure the right trade-off between inefficiency and synchronization overhead.

5.4.3 Overhead of State Distribution

Table 5.3 portrays the incurred message load in the C2C communication for the transmission of state-updates resulting from 1000 subsequent SDN-LB mapping requests, distributed uniformly across all instances. The portrayed result considers the average *per-instance* overhead in a cluster of 5 replicas.

The AC's Batched-Mode incurs the lowest message overhead due to its useful property of state-update aggregation. The SC mode depicts a lower number of frames transmitted during the per-second measurement intervals. However, the average frame size of the SC-transmitted messages is larger compared to AC and EC deployments. The total time taken to serve 1000 SDN-LB embeddings with SC takes longer due to each write *and* read request requiring serialization. Lastly, AC's distribution time suffers compared to EC, since EC processes transactions as fast as possible and does not implement the overhead of consistency adaptation.

Consistency Model	Distr. Time $[s]$	Avg. PPS	Avg. Packet Size $[B]$	Avg. Load $[B/s]$
EC	25.12	3802	124.5	469k
SC	115.12	326.82	2612.6B	844.2k
AC (Batched-Mode)	39.501	3460	105.4	365k
AC (Fast-Mode)	40	3552	102.5	372k

Table 5.3: Per-replica load when serving 1000 SDN-LB requests in a 5-controller cluster synchronized using SC, AC or EC.

5.5 Chapter Summary and Outlook

This chapter presented the realization of an AC framework. On an example of a 5-controller SDN cluster and two realistic network topologies, we highlighted its advantages w.r.t.: i) the control plane response time compared to the SC approach; ii) the decision-making efficiency compared to the EC approach; iii) the generated C2C load compared to both SC and EC.

We have introduced two distribution abstractions that enable the C2C state exchange, while minimizing the response time and the generated average load. Furthermore, we have presented two adaptation functions that adapt the system in a closed-loop manner, so that the target inefficiency incurred by the eventuality of state delivery persists according to the SDN application's expectations.

AC-based distributed SDN control plane paves the way for scalable control plane designs. Enabling non-synchronized data plane configurations is especially relevant for systems that require fast response (i.e., on-demand TSN stream establishment). By not relying on costly consensus on each resource state update, end-clients profit from shortened request-handling time. If state synchronization conflicts occur and correctness is endangered, our system adapts autonomously to a more appropriate CL.

Open points: Future iterations of our work should evaluate the AC framework in scenarios comprising larger number of controllers. Large-scale demonstrations may further emphasize the benefits over the alternative consistency approaches. Additionally, the introduced adaptation functions take as input the SDN application-triggered inefficiency reports. The benefit of the consistency adaptation, however, comes at the expense of an expanded model parametrization space and the necessity of an SDN application re-design. Further attention should hence be put on simplifying the related development efforts, e.g., by automating the generation of "good" parametrizations using established parameter tuning methods [32, 205].

Chapter 6

Enabling Fault Tolerance in Byzantine Fault Tolerant Control Plane

The Fail-Stop approaches discussed in previous chapters rely on the assumption of correct decision-making in the controller replicas. Successful introduction of SDN in the critical industrial networks, however, also requires catering to the issue of *unreliable* (e.g., buggy) and / or *malicious* controller failures [179] (ref. Section 2.2.1). We identify the faults resulting in these types of failures as *Byzantine* faults and classify the designs tolerating these as BFT approaches.

We henceforth propose MORPH, a distributed SDN control plane capable of tolerating Fail-Stop and Byzantine failures, that distinguishes and localizes faulty controller instances and accordingly dynamically reconfigures the control plane. Furthermore, we present an Integer Linear Program (ILP) formulation of the corresponding C2S connection assignment problem that leverages the awareness of the source of failure (i.e., Fail-Stop or Byzantine-induced), in order to optimize the number of active controllers and minimize the C2C and C2S reconfiguration delays.

The proposed re-assignment executes dynamically after each successful failure identification. $2F_M + F_A + 1$ controllers are required to tolerate F_M Byzantine and F_A availability-induced failures. On each successful detection of F_M Byzantine controllers, the proposed design reconfigures the control plane to require a *single* controller message to forward the system state.

We evaluate MORPH in an emulated testbed comprising up to 16 controller replicas and up to 34 switches, so to tolerate up to 5 unique Byzantine and additional 5 availability-induced controller failures (a total of 10 unique controller failures). We quantify and highlight the dynamic decrease in the packet and CPU load and the response time after each successful failure detection. The demonstrated scenario focuses on a resource-aware routing application replicated among MORPH controller instances.

The proposed approach relies on BFT assignment of sequence numbers for each incoming request. The sequencing of the requests is done in order to guarantee serialization of request execution in each controller replica. The sequencing procedure takes place during the C2C interaction prior to computation of the outputs of requests. In this chapter, we provide the general design of a distributed control plane capable of tolerating Byzantine failures, the distributed client request execution, request output comparison in configuration targets, identification of Byzantine instances and subsequent control

plane reconfiguration. We detail the particularities surrounding efficient C2C interactions required for sequencing of client requests in the subsequent Chapter 7.

The proposed MORPH design and its analysis were published in augmented form in [1].

6.1 Related Work - On Enabling BFT in Context of SDN

Prior to introducing MORPH, we henceforth give a brief overview of existing research on enabling BFT in context of SDN.

The prominent open-source SDN platforms such as ODL [123] and ONOS [39] support no means of identifying Byzantine failures. Indeed, the single-leader Raft is susceptible to Byzantine failures. The Raft *leader* processes and broadcasts any new data store updates to the *followers* before an update may be considered committed. This introduces multiple attack vectors, e.g.: i) the adversary may generate malicious state updates in each follower if the leader controller is in the possession of a Byzantine adversary; ii) inconsistencies in the state view of master and followers if adversary takes over the Raft follower [113]; iii) incorrectness issues, e.g., a continuous non-convergence of controller cluster leadership [201].

The definition of the problem of unreliable SDN controller instances, including an initial solution draft, was proposed in the security context in [115]. The authors discuss the requirement of a minimum assignment of *3F+1* controllers to each switch *at all times*, so to tolerate a total number of *F* Byzantine failures. *3F+1* matching messages are required during the *agreement* and *2F+1* during the *execution* phase in order to reach the majority and consensus on the correct response after the comparison of the computed configuration outputs [197].

BFT-SMaRt [41] proposes a strong consistent framework for supporting Byzantine- and Fail-Stop failures. The authors abstract away the notion of a "failure" to consider controllers *failed*, if either the controller replica crashes and never recovers or if the replica continues repeatedly crashing and recovering. Their follow-up work [47] applies the *Bft-SMaRt* in order to improve the data store replication performance in a strong consistent SDN. They evaluate the workload generated by real SDN applications as they interact with the data store.

Both works do not consider the issue of C2C synchronization, where an adversary replica may initiate malicious state synchronization procedure and thus commit malicious database changes in the correct controllers. Furthermore, they do not cover for the aspects of an adaptive C2S connection reassignment procedure nor do they distinguish and leverage the difference between Byzantine and Fail-Stop failures.

A recent proposal for a BFT-enabled SDN [128] advocates the usage of a total of *2F+1* instead of *3F+1* controller instances. The authors propose the collection of *F+1* PRIMARY controller configuration messages at each switch (generated by their PRIMARY controllers), and a delayed request to the remaining *F* SECONDARY instances if an inconsistency is detected in the switch. The configuration message is flagged as correct only after a minimum of *F+1* matching configurations are

received in the switch. The authors do not consider the impact of source of failure nor the impact of their proposed design on the correctness of the C2C state synchronization process.

6.2 MORPH: A Design Enabling Byzantine Fault Tolerant Operation

6.2.1 Terminology

We rely on the following terminology to introduce MORPH:

- A *Byzantine controller* is an active but unreliable controller instance. It may disguise itself as a *correct* controller for an arbitrary period of time. Byzantine controllers may compute an incorrect result as a consequence of e.g., a buggy internal state or an adversary takeover. Until suspected, all Byzantine controllers are considered *correct*.
- A *correct controller* is an active controller instance which is not declared *Byzantine* and which actively participates in the cluster.
- An *incorrect controller* is a controller that is either declared Byzantine or is *unavailable*.
- An *unavailable controller* is an inactive controller as a result of a hardware / software failure. It is unknown if an unavailable controller was either *correct* or *Byzantine* at some point in time prior to its failure.

6.2.2 System assumptions

We extend the typical SDN control plane with additional functional elements to enable a BFT operation. The control plane communication between the switches and controllers, i.e., S2C, C2S; and between controllers, i.e., C2C, assumes an IBC channel [169]. Furthermore, all communication between its elements (i.e., `REASSIGNER`, `C-COMPARATOR` and `S-COMPARATOR`) is assumed to be signed [148]. Thus *i) message forging is assumed impossible* and *ii) message integrity is ensured in the data plane during normal operation*. In order to prevent a faulty replica from impersonating a correct replica, correct replicas can authenticate each message using Message Authentication Code (MAC)-based authentication [53, 65].

6.2.3 Overview of Goals of MORPH

MORPH accomplishes two major objectives, *i) data plane protection* from Byzantine controllers and *ii) re-adaptation to the optimal network configuration* w.r.t. C2S connection assignments based on the present number of correct and incorrect controllers:

- *Data plane protection*: Even if there are up to F_M Byzantine controllers and F_A failed controllers in the network, the data plane elements must *never* be incorrectly reconfigured or tampered with by the incorrect controllers. We realize this protection by leveraging *replicated computation* of control plane decisions and their transmission from multiple controllers to the target switches.

Hence, each switch is connected to multiple controllers at the same time, and only accepts and proceeds to commit the reconfiguration requests if a sufficient number of matching messages for reaching the *correct consensus* was received. If we consider a scenario where F_M Byzantine controllers send their reconfiguration messages to switches, it becomes clear that we need at least $F_M + 1$ matching reconfiguration messages to reach *correct consensus* and for the switches to distinguish a correct reconfiguration request from an *inconsistent* message set.

For the detection of an unavailability-induced controller failure (i.e., a Fail-Stop failures), we deploy the ϕ-FD failure detector [79, 80] in switches that reliably identifies the failed, non-Byzantine controllers [49]. Hence, in the case of F_A Fail-Stop controller failures, we only require one additional backup instance (i.e., $F_A + 1$) to achieve a correct control plane protection. Therefore, in order to protect the data plane from F_M Byzantine controllers and F_A failed controllers, each switch has to be assigned at least $2F_M + F_A + 1$ controllers. Granted, a switch will accept the new reconfiguration request already after receiving $F_M + 1$ matching messages from the controllers.

Additionally, individual switch failures may cause packet loss and thus an unsuccessful delivery of controller messages. If a faulty switch on the control path starts dropping controllers' messages, the next-in-line switches and dislocated controllers eventually start suspecting failed connections to the unreachable controllers. The switches eventually raise an alarm at the `REASSIGNER`. The `REASSIGNER` then accordingly marks the unreachable controllers as *unavailable*. F_A unavailable controllers are tolerated by the design. Thus, both failures coming from unavailable controllers, as well as from the unavailable switches that forward the control packets to / from unreachable controllers are accounted by F_A.

- *Adaptation to the optimal network configuration*: After the detection of an *incorrect* controller, the *tamper-proof* entity `REASSIGNER` recomputes C2S connection assignments in order to achieve the optimal network configuration. The recomputed assignments are distributed to the controllers and switches in order to reduce the reconfiguration delay and messaging overhead in the control plane. The definition of the optimal configuration of a network and the reassignment logic is elaborated in Section 6.3.2.

6.2.4 Architecture for Tolerating Byzantine Failures

The following elements are introduced in the MORPH system architecture depicted in Fig. 6.1:

Northbound Interface (NBI) and data plane clients: The SDN application clients requesting for a particular controller service. NBI clients are aware of the controllers but not of their roles w.r.t. switch assignment. Thus, NBI clients report their requests to all available controllers. The data plane clients feed their requests to a well-defined controller group address. The data plane client requests are then handled by the neighboring edge switch and are encapsulated as a `PACKET-IN` message (e.g., using OF [122]) and delivered only to the PRIMARY and SECONDARY controllers of that switch.

`S-COMPARATOR`: A switch mechanism that collects and compares the configuration outputs generated at each active PRIMARY or SECONDARY controller instance. On discovery of inconsistencies,

the S-COMPARATOR reports the conflicting configurations to the REASSIGNER. We assume a tamper-proof operation of the S-COMPARATOR (e.g., realized using the Intel Software Guard Extensions (SGX) enclaves [211]). S-COMPARATOR cannot be implemented with current OF logic, but requires an additional software agent, responsible for comparison of arriving control messages. This requirement is, however, not unique to MORPH and holds for any BFT-enabled system where switches take over the comparison logic, including [128] and [115]. A running instance of S-COMPARATOR is assumed in each active switch (refer to Fig. 6.1). Please note that the S-COMPARATOR does not necessarily imply a user-space software agent relying on the switch's general-purpose CPU. Advances in programmable switching hardware [46] can enable hardware-accelerated implementations of the S-COMPARATOR. Our take on a data plane accelerated S-COMPARATOR is discussed in the next chapter.

C-COMPARATOR: A controller mechanism that collects and compares the controller configuration messages generated at each active controller replica. In the replica hosting stateful applications (ref. Section 6.2.5), it withholds the propagation of switch configuration messages as long as the majority of controller updates remains inconsistent. After collecting a majority of consistent messages, it updates the underlying controller's internal state configuration to reflect the configuration update (e.g., it updates the status of resource reservations). A running instance of C-COMPARATOR is assumed in each active controller replica (ref. Fig. 6.1).

REASSIGNER: The REASSIGNER is in charge of detecting the incorrect controllers. It is furthermore able to find an optimal C2S assignment so to exclude the incorrect controllers and report the updated C2S list assignments at the controllers and switches. The REASSIGNER is triggered every time a controller is suspected by an S-COMPARATOR. The trigger is executed independent of the root cause of failure (i.e., the discovery of a Byzantine or unavailable controller). However, the detected root cause of failure impacts the selection of controllers considered in the assignment. The proposed C2S assignment solver is aware of the controllers' capacities and the worst-case C2S and C2C delays, and is hence able to take into account the worst-case bounds when executing an optimal assignment. We detail the assignment procedure in Section 6.3.2.

Note: We cannot trust a single REASSIGNER instance to be correct. To make the REASSIGNER resistant against Byzantine and availability failures, similarly to controllers, it must be implemented as a group of replicas that carry out a BFT agreement step after each successful reassignment. For brevity, in the rest of the paper, we assume a tamper-proof operation of the REASSIGNER (e.g., realized using the SGX enclaves [211]).

6.2.5 Distinguishing Application Types

We distinguish two SDN application types: State Independent Applications (SIAs) and State Dependent Applications (SDAs). SIAs refer to all SDN applications which process client requests independent of the actual network state. Hence, given a repeated client input, a correct SIA repeatedly generates a constant, semantically equal response. A representative of SIA is the *hop-based shortest path routing* where, assuming no transient topology changes, the shortest path between the hosts remains the same, regardless of the current link and node utilization.

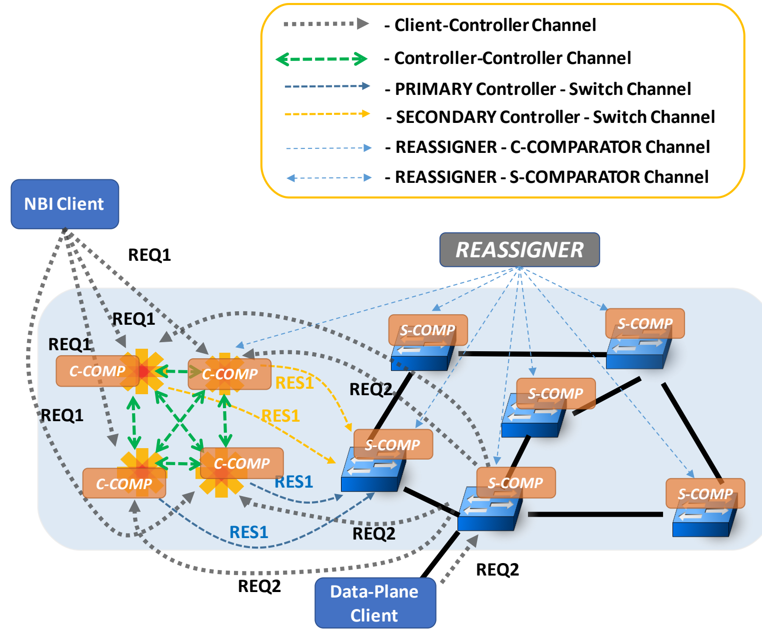


Figure 6.1: MORPH architecture comprising the: i) SDN controllers that host the C-COMPARATOR; ii) switches that host the S-COMPARATOR; iii) NBI and data plane clients; and iv) REASSIGNER element. REASSIGNER is in charge of dynamically recomputing the C2S assignments after a reported controller failure discovery by S-COMPARATOR.

In contrast to SIA, SDA refers to all applications which base their decision-making on the current network state. Hence, if we consider typical *load balancing routing*, two identical requests (e .g., path requests) could generate completely different responses (i.e., different routes) from the controller, so to optimize for the total *current* resource utilization given the instantaneous network state.

6.2.6 Necessity of Controller-to-Controller Synchronization

Consensus across SDA instances: In a resource-based SDA, such as in a path reservation application, the weight of each link may depend on existing reservations. There, a consistency issue may arise if multiple clients concurrently request new embeddings. Due to propagation and processing delays, different controller replicas may process the corresponding requests in a *non-deterministic order* and thus produce inconsistent responses to switches. Thus, a C2C synchronization step is necessary to agree on the correct message output.

MORPH realized this step *after* the execution of request. Namely, propagation of configuration messages to switches is delayed until a sufficient number of matching controller messages is collected among the controller replicas. In order to achieve the consensus: i) all PRIMARY and SECONDARY controllers first compute a response immediately after receiving the client request; ii) they next store it in an internal database and; iii) exchange their decisions using the any-to-any C2C channels. Finally, the *consensus* is reached in the C-COMPARATOR when it receives at least $\lfloor (Req_P + Req_S)/2 \rfloor + 1$ identical responses for a given request, where Req_P is the current number of required PRIMARY controllers and Req_S is the current number of required SECONDARY controller assignments per switch.

Note: In Chapter 7, we investigate the response time advantages of controller replicas synchronizing on the order of request execution *prior* to executing the requests. Thus, post-synchronization step described here then becomes unnecessary.

Proxying mechanism in SDA and SIA instances: Both in the SDA and SIA routing applications, the establishment of certain long paths is not achievable without C2C synchronization. For example, a client could initiate a routing request via its neighboring edge switch S_A to the set of S_A's assigned PRIMARY and SECONDARY controllers. However, if the determined path contains a switch which is not assigned to the same set of PRIMARY and SECONDARY controllers as with S_A, the configuration of the flow rules by the computing controller would not be allowed. Therefore, when a controller assigned the role of PRIMARY or SECONDARY controller for the target switch compares and validates the new configuration on the C2C channel, it dispatches a response to the affected switches.

6.3 Detailed MORPH Design

Next we detail the system flow and the algorithms behind MORPH. In the second part, we formulate the objectives and constraints for the ILP that dynamically reassigns the C2S connections at runtime, as part of the `REASSIGNER` logic.

6.3.1 Algorithm Specification

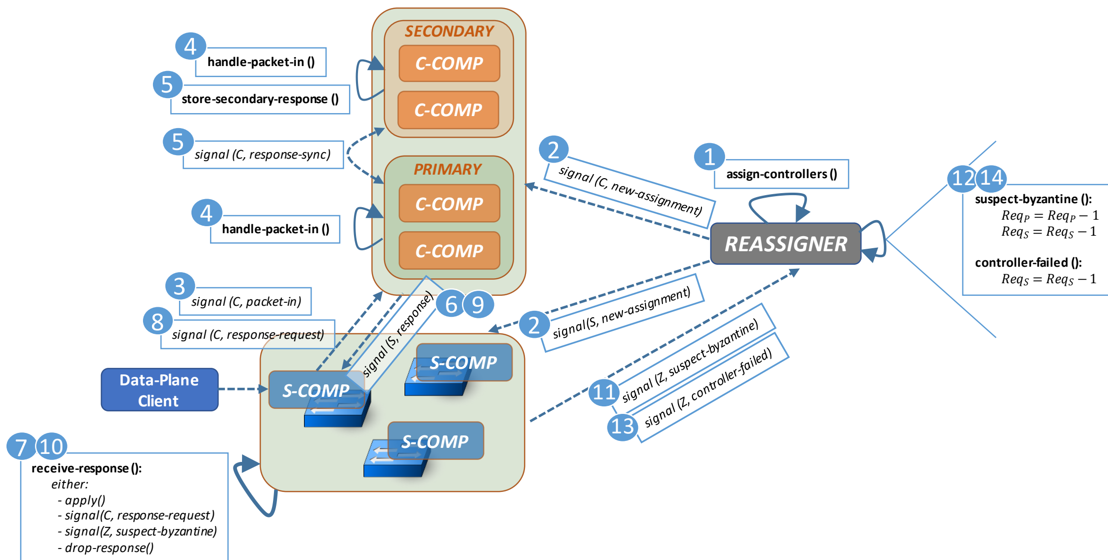


Figure 6.2: A simplified visual representation of MORPH's workflow and the distributed algorithm execution. The portrayed workflow depicts the steps of: ((1)-(2)) controller-switch assignment; ((3)-(5)) client request dissemination and handling in the PRIMARY and SECONDARY `C-COMPARATOR` instances; ((6)-(7), (8)-(10)) dissemination and handling of PRIMARY and SECONDARY controller responses in the affected `S-COMPARATOR` instances, respectively; ((11)-(12), (13)-(14)) dissemination and handling of Byzantine and / or Fail-Stop failures in the `REASSIGNER`, respectively.

The attached algorithms describe the operation of MORPH in more detail. A visual overview of introduced algorithms is given in Fig. 6.2. We distinguish the following steps:

1. Given a list of controllers and switches, as well as the list of clients and their worst-case capacity requirements, `REASSIGNER` executes the initial assignment of C2S connections. It considers the unidirectional delay as well as the controller capacity constraint. As described in Section 6.3.2, the `REASSIGNER` minimizes the total number of active controllers when assigning switches to controllers. This process is embodied in Lines 1-2 of Alg. 6.

2. The C2S assignment lists are distributed to the SDN controllers and switches. Switches are thus assigned their PRIMARY and SECONDARY controllers. From now on, any received configuration message initiated by a remote controller is queued for the evaluation in the switch if and only if the configuration message was initiated by a controller that belongs to either the PRIMARY or SECONDARY controller set of that switch. Otherwise, the message is dropped.

3. An end-client sends off their request to the SDN controller, i.e., a request for a computation of a constrained path, a load-balancing request etc. NBI clients send their requests directly to all controllers, while data plane clients stay unaware of the location of the controllers, so to simplify the client-side / user logic [165]. The data plane clients feed their requests directly into the network. The next-hop edge switch then intercepts and proxies the request to its assigned PRIMARY and SECONDARY controllers.

4. The PRIMARY and SECONDARY controllers of the switch which initiated the client request compute the corresponding configuration response and decide to apply the computed response configuration in the affected switches. We enable the selection of a flexible trade-off between the switch configuration time overhead and the generated control plane load. Replicas assigned to a switch are either of PRIMARY or SECONDARY role. Depending on the point in time the SECONDARY replicas decide to forward the computed configuration responses to their switches, the overhead in terms of response time and control plane load varies. Therefore, we define the following two models:

   - *NON-SELECTIVE*: Propagation of computed configurations from SECONDARY controllers to switches initiates immediately after receiving the client request. In case of the SDA, new configurations are sent to switches after reaching the majority consensus on the final configuration message, as explained in Sec. 6.2.6.

   - *SELECTIVE* model imposes an additional step of buffering the intermediate result (either locally computed in the case of SIA or the majority result in the case of SDA) and forwarding the result to the switch on-demand, whenever the switch requests additional configuration messages from its SECONDARY controllers.

   In the case of the *SELECTIVE* model, only the controllers which are the PRIMARY controllers of the to-be-configured switches forward their request directly to the switch. In the case of the *NON-SELECTIVE* model, the SECONDARY controllers also proactively forward their configuration

messages to the switches. This differentiation is depicted in Lines 9-16 of the SIA-specific Alg. 7 and Lines 11-18 of the SDA-specific Alg. 8.

State Independent Application (SIA): The controllers forward the computed configuration messages to the target switches if they are assigned either the PRIMARY or SECONDARY role for the target switch. If the controller that computes the configuration is assigned neither role for the target switch, the configuration result is forwarded to the switch only after a consensus is achieved on the actual PRIMARY / SECONDARY controllers of the switch. For SIA, consensus is achieved after collecting Req_P identical messages on the PRIMARY / SECONDARY instances. Req_P denotes the currently required number of assigned PRIMARY controllers per switch.

State Dependent Application (SDA): In case of SDA, all correct controllers must first reach consensus on their common internal state update. To this end, they deduce the majority configuration message (in Lines 8-9 of Alg. 8). Majority configuration response is necessary in order to omit the possibility of false positive detection, where *correct* controllers potentially become identified as faulty instances, following a temporary controller state de-synchronization during runtime, as discussed in Section 6.2.6. After determining the majority response, the controllers send the configuration message to the switches (as in Lines 10-16 of Alg. 8).

5. The `S-COMPARATORS` collect the configuration requests and decide after Req_P matching PRIMARY messages to apply the configuration locally. In the SELECTIVE scenario where inconsistent messages among the Req_P PRIMARY messages are detected, the switches contact the SECONDARY replicas to fetch additional Req_S responses - as depicted in Line 10 of Alg. 9. After receiving Req_P consistent (equal) messages (Line 12-15 of Alg. 9), the switches apply the new configuration (Line 15). Thus, the overhead of collection of Req_P consistent messages dictates the worst-case for applying new switch configurations.

6. After discovery of an inconsistency among the controller responses (Lines 9-10 of Alg. 9) or a failure of an assigned controller (Line 24-25), the `S-COMPARATORS` wait until $Req_P + Req_S$ messages are collected, before addressing the `REASSIGNER` with the controller identifiers and the conflicting messages (Line 20 of Alg. 9). If a replica has failed, the duration of the time the switch waits before contacting the `REASSIGNER` corresponds to the worst-case failure detection period (dictated by the underlying failure detector, e.g. the ϕ-FD [80]). If a switch suspects that a its assigned replica has failed, it notifies the `REASSIGNER` as per Line 25 of Alg. 9 and Line 13 of Alg. 6.

7. The `REASSIGNER` compares the inputs and deduces the majority of correct responses received for the conflicting client request. If a Byzantine failure of a controller is suspected, both Req_P and Req_S are decremented by one for each Byzantine replica. If a replica is marked as unavailable (independent of the fault type), only the required number of SECONDARY controllers per-switch Req_S is decremented by one. Lines 4-17 of Alg. 6 contain this differentiation.

8. Based on the updated Req_P and Req_S controllers deduced during runtime, the `REASSIGNER` computes the new optimal assignment and configures the switches and controllers with the new

C2S assignment lists. Steps 3-7 repeat until all F_M Byzantine and F_A unavailable controllers are eventually identified.

By lowering the number of maximum required PRIMARY and SECONDARY controller assignments per switch, the `REASSIGNER` minimizes the total control plane overhead in terms of the packet exchange in both C2C and C2S channels, as well as the time required to confirm new controller and switch state configurations.

Algorithm 6 `REASSIGNER`: Controller-switch assignment.

Notation:
$\mathcal{S}$ Set of available SDN switches
C Set of available SDN controllers
$\mathcal{B}$ Set of detected blacklisted SDN controllers
$\mathcal{A}_p$ Set of PRIMARY C2S assignments
$\mathcal{A}_s$ Set of SECONDARY C2S assignments
Req_P No. of required PRIMARY controllers per switch
Req_S No. of required SECONDARY controllers per switch
F_M Maximum number of tolerated Byzantine failures
F_A Maximum number of tolerated Fail-Stop failures

Initial variables:
$Req_P = F_M + 1$
$Req_S = F_A + F_M$

1: **procedure** CONTROLLER-SWITCH ASSIGNMENT
2: $(\mathcal{A}_p, \mathcal{A}_s) :=$ *Assign-Controllers* $(\mathcal{S}, C)$
3:
4: **upon event** *suspect-byzantine* $< C_m >$ **do**
5: $C \leftarrow C \setminus C_m$
6: $\mathcal{B} \leftarrow \mathcal{B} \cup C_m$
7: $Req_P = Req_P - 1$
8: $Req_S = Req_S - 1$
9: $(\mathcal{A}_p, \mathcal{A}_s) :=$ *Assign-Controllers* $(\mathcal{S}, C, Req_P, Req_S)$
10: *signal* $(\mathcal{S}$, *new-assignment* $< \mathcal{A}_p, \mathcal{A}_s >)$
11: *signal* $(C$, *new-assignment* $< \mathcal{A}_p, \mathcal{A}_s, \mathcal{B} >)$
12:
13: **upon event** *controller-failed* $< C_f >$ **do**
14: $C \leftarrow C \setminus C_f$
15: $\mathcal{B} \leftarrow \mathcal{B} \cup C_m$
16: $Req_S = Req_S - 1$
17: $(\mathcal{A}_p, \mathcal{A}_s) :=$ *Assign-Controllers* $(\mathcal{S}, C, Req_P, Req_S)$
18: *signal* $(\mathcal{S}$, *new-assignment* $< \mathcal{A}_p, \mathcal{A}_s >)$
19: *signal* $(C$, *new-assignment* $< \mathcal{A}_p, \mathcal{A}_s, \mathcal{B} >)$

6.3.2 Optimized Controller-Switch Assignment

`REASSIGNER` assigns the controller roles to switches on a per-switch basis. It takes a list of controllers and switches, their unidirectional delay and delay requirements, as well as the list of clients and the client request arrival rates as input.

For brevity, we henceforth define a single objective for the assignment problem - the `REASSIGNER` minimizes the number of active controllers, so to lower the total overhead of C2C communication, while taking into consideration the maximum delay bound and available controller capacities when assigning

Algorithm 7 `C-COMPARATOR`: Handling SIA requests in the SDN controller.

Notation:
P Client request (e.g., flow request) initiated at switch S_C
C The set of available SDN controllers
C_i The local controller instance
C_r The remote controller instance
R_{P,S_i} Configuration response intended for switch S_i
$\mathcal{A}_p$ Set of primary C2S assignments
$\mathcal{A}_s$ Set of secondary C2S assignments
A_{S_i,C_i} Connection assignment variable for C2S pair (S_i, C_i)

```
1:  procedure HANDLE CLIENT REQUEST
2:   upon event packet-in < S_C, P > do
3:      R_P := handle-packet-in (P)
4:      signal (C, response-sync< C_i, R_P >)
5:
6:   upon event  response-sync < C_r, R_P > do
7:      if is-sia-consensus-reached(R^P) or C_i == C_r then
8:          for all S_i ∈ affected-switches(R_P) do
9:              if mode == non-selective then
10:                 if A_{S_i,C_i} ∈ A_p ∪ A_s then
11:                     signal (S_i, response < C_i, R_{P,S_i} >)
12:             else if mode == selective then
13:                 if A_{S_i,C_i} ∈ A_p then
14:                     signal (S_i, response < C_i, R_{P,S_i} >)
15:                 else if A_{S_i,C_i} ∈ A_s then
16:                     store-secondary-response(R_{P,S_i})
17:
18:  upon event  response-request < S_i, P > do
19:     signal (S_i, response < C_i, R_{P,S_i} >)
```

controllers to switches. The mentioned objective additionally allows for implicit load-balancing of C2S connections, as long as all controllers are instantiated with an equal initial capacity.

Let $A^P_{C_i,S_j}$ and $A^S_{C_i,S_j}$ denote the active assignment of a controller instance C_i to the switch S_j as a PRIMARY / SECONDARY controller, respectively. Then U_{C_i} denotes the active participation of the controller C_i in the system:

$$U_{C_i} = \begin{cases} 1 & \text{when } \sum\limits_{S_j \in S} A^P_{C_i,S_j} + \sum\limits_{S_j \in S} A^S_{C_i,S_j} > 0, \forall C_i \in C \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

The objective function can then be formalized as:

$$min \sum_{C_i \in C} U_{C_i} \tag{6.2}$$

In the following, we define the constraints of the corresponding ILP.

Minimum Assignment Constraint:

In order to tolerate F_M Byzantine and F_A unavailability failures, initially at time $t = 0$ the `REASSIGNER` assigns $Req_P(0) = F_M + 1$ PRIMARY controllers, and $Req_S(0) = F_M + F_A$ SEC-

Algorithm 8 `C-COMPARATOR`: Handling of SDA requests in the SDN controller.

Notation:
P Client request (e.g. flow request) initiated at switch S_C
C The set of available SDN controllers
C_i The local controller instance
C_r The remote controller instance
R^{maj}_{P,S_i} Majority configuration for switch S_i
$\mathcal{R}^P$ Buffer containing controller responses for request P
$\mathcal{A}_p$ Set of primary C2S assignments
$\mathcal{A}_s$ Set of secondary C2S assignments
A_{S_i,C_i} Connection assignment variable for C2S pair (S_i, C_i)

1: **procedure** HANDLE CLIENT REQUEST
2: **upon event** *packet-in* $< S_C, P >$ **do**
3: R_P := *handle-packet-in* (P)
4: *signal* (C, *response-sync*$< C_i, R_P >$)
5:
6: **upon event** *response-sync* $< C_r, R_P >$ **do**
7: $\mathcal{R}^P \leftarrow \mathcal{R}^P \cup < C_r, R_P >$
8: **if** *is-sda-consensus-reached*($\mathcal{R}^P$) **then**
9: R^{maj}_P :=*majority-response*($\mathcal{R}^P$)
10: **for all** $S_i \in$ *affected-switches*(R^{maj}_P) **do**
11: **if** mode == *non-selective* **then**
12: **if** $A_{S_i,C_i} \in \mathcal{A}_p \cup \mathcal{A}_s$ **then**
13: $signal\ (S_i, response < C_i, R^{maj}_{P,S_i} >)$
14: **else if** mode == *selective* **then**
15: **if** $A_{S_i,C_i} \in \mathcal{A}_p$ **then**
16: $signal\ (S_i, response < C_i, R^{maj}_{P,S_i} >)$
17: **else if** $A_{S_i,C_i} \in \mathcal{A}_s$ **then**
18: *store-secondary-response*(R^{maj}_{P,S_i})
19:
20: **upon event** *response-request* $< S_i, P >$ **do**
21: $signal\ (S_i, response < C_i, R^{maj}_{P,S_i} >)$

ONDARY controllers per each switch. The values Req_P and Req_S are time-variant and are adapted on every discovered Byzantine / Fail-Stop failure or a successful controller recovery:

$$\begin{aligned} \sum_{C_i \in C} A^P_{C_i,S_j} \geq Req_P(t), \forall S_j \in \mathcal{S} \\ \sum_{C_i \in C} A^S_{C_i,S_j} \geq Req_S(t), \forall S_j \in \mathcal{S} \end{aligned} \tag{6.3}$$

Unique Assignment Constraint: Each controller C_i may either be assigned the role of a PRIMARY *or* SECONDARY, at maximum *once* per switch S_j:

$$A^P_{C_i,S_j} + A^S_{C_i,S_j} \leq 1, \forall C_i \in C, S_j \in \mathcal{S} \tag{6.4}$$

Controller Capacity Constraint: Let P_{C_i} denote the total available controller's C_i capacity. Let L_{CL_k} and L_{S_j} denote the load stemming from the NBI client CL_k and the edge switch S_j, respectively. The sum of the loads generated by the assigned switches at controller C_i may not exceed the *difference*

Algorithm 9 `S-COMPARATOR`: Processing of controller configuration messages in the switch.

Notation:
Req_P No. of required PRIMARY controllers per switch
Req_S No. of required SECONDARY controllers per switch
$\mathcal{R}^P$ The set of received responses for client request P
$\mathcal{A}_p^{S_i}$ Set of PRIMARY controllers assigned to switch S_i
$\mathcal{A}_s^{S_i}$ Set of SECONDARY controllers assigned to S_i
$\mathcal{Z}$ The set of designated shufflers assigned to switch S_i

1: **procedure** COMPARE AND APPLY CONTROLLER CONFIGURATIONS
2: **upon event** *receive-response*$< C_j, R^P >$ **do**
3: $\mathcal{R}^P \leftarrow \mathcal{R}^P \cup < C_j, R^P >$
4: **if** $C_j \in \mathcal{A}_p^{S_i}$ **then**
5: **if** $|\mathcal{R}^P| == Req_P$ **then**
6: $\mathcal{R}_{majority}$:= *majority-responses*$(\mathcal{R}^P)$
7: **if** $|\mathcal{R}_{majority}| \geq Req_P$ **then**
8: *apply* $(\mathcal{R}_{majority})$
9: **else**
10: *signal*$(\mathcal{A}_s^{S_i}$, *response-request* $< S_i, P >)$
11: **else if** $C_j \in \mathcal{A}_p^{S_i} \cup A_s^{S_i}$ **then**
12: **if** $Req_P < |\mathcal{R}^P| \leq Req_P + Req_S$ **then**
13: $\mathcal{R}_{majority}$:= *majority-responses*$(\mathcal{R}^P)$
14: **if** $|\mathcal{R}_{majority}| \geq Req_P$ **then**
15: *apply* $(\mathcal{R}_{majority})$
16: **for all** $< C_i, R_i > \in \mathcal{R}^P$ **do**
17: **if** $R_i \notin \mathcal{R}_{majority}$ **then**
18: $\mathcal{C}_{incnst} \leftarrow \mathcal{C}_{incnst} \cup C_i$
19: **if** $C_i \neq \emptyset$ **then**
20: *signal* $(\mathcal{Z}$, *suspect-byzant* $< \mathcal{C}_{incnst} >)$
21: **else**
22: *drop-response*(R^P)
23:
24: **upon event** *controller-failed*$< C_j >$ **do**
25: *signal* $(\mathcal{Z}$, *controller-failed* $< C_j >)$

of the *total available controller's capacity* and the *sum of the constant loads* stemming from the NBI clients:

$$\sum_{S_j \in \mathcal{S}} A_{C_i,S_j}^P \cdot L_{S_j} + \sum_{S_j \in \mathcal{S}} A_{C_i,S_j}^S \cdot L_{S_j} \leq P_{C_i} - \sum_{CL_k \in \mathcal{CL}} L_{CL_k}, \forall C_i \in \mathcal{C} \tag{6.5}$$

Delay Bound Constraint: We assume well-defined worst-case upper bounds for the unidirectional delays in C2S and C2C communication. Let d_{C_i,S_j} and d_{C_i,C_j} denote the guaranteed worst-case experienced delay for the C2S pair (C_i, S_j) and C2C pair (C_i, C_j), respectively. Given a maximum tolerable global upper bound delays $D_{C,S}$ and $D_{C,C}$, we define the related constraints as:

$$\begin{aligned} A_{C_i,S_j}^P \cdot d_{C_i,S_j} &\leq D_{C,S}, \forall C_i \in \mathcal{C}, S_j \in \mathcal{S} \\ A_{C_i,S_j}^S \cdot d_{C_i,S_j} &\leq D_{C,S}, \forall C_i \in \mathcal{C}, S_j \in \mathcal{S} \\ d_{C_i,C_j} &\leq D_{C,C}, \forall C_i \in \mathcal{C} \setminus C_j, C_j \in \mathcal{C} \setminus C_i \end{aligned} \tag{6.6}$$

Table 6.1: Notation used in specification of MORPH's communication overhead.

Notation	Property
Client-to-Switch (CLI2S)	Communication channels between clients and switches
S2C	Upstream communication channels between switches and controllers
C2C	Controller-to-Controller communication channels
C2S	Downstream communication channels between controllers and switches
m	Total number of data plane clients in the network
μ_{cli}	The average request rate of data plane clients
$\lvert S_{cli} \rvert$	The average number of affected switches by data plane client's requests
$\lvert C \rvert$	The total number of controllers, $\lvert C \rvert \geq 2F_M + F_A + 1$
$f_{M,A}$	The current number of failed controllers
E	Equal to 1/0 if MORPH is in NON-SELECTIVE / SELECTIVE mode

6.3.3 Communication and Computation Complexity

We henceforth analyze the communication overhead imposed by MORPH framework when handling data plane clients only (NBI clients excluded for brevity). Multiple communication channels are present in the system:

- Client-to-Switch (CLI2S): Every request generated by a data plane client is first forwarded to the corresponding edge switch, therefore, if the average request generation rate of m data plane clients is μ_{cli}, then $m \cdot \mu_{cli}$ is the total request generation rate on CLI2S channels.

- Switch-to-Controller (S2C): Upon receiving the request, the edge switch forwards it to its PRIMARY and SECONDARY controllers, hence, the total message rate on the *upstream* communication channels between *switches* and *controllers* is $(Req_P + Req_S) \cdot m \cdot \mu_{cli}$. In the best-case, all faulty controllers were detected, hence the total number of PRIMARY and SECONDARY controllers is reduced to $Req_P = 1$ and $Req_S = 0$, while in the worst-case none of the faulty controllers failed yet, i.e., the inital values still hold $Req_P = F_M + 1$ and $Req_S = F_M + F_A$.

- Controller-To-Controller (C2C): For every request received from the switches, its PRIMARY and SECONDARY controllers calculate the corresponding response (i.e., a forwarding path) and forward it to all of the *other available* (non-faulty) controllers in the network (i.e., $|C| - f_{M,A} - 1$). The total dynamic rate rate on the C2C channel is $(|C| - f_{M,A} - 1) \cdot (Req_P + Req_S) \cdot m \cdot \mu_{cli}$. In the best-case, all failed controllers were detected, thus $f_{M,A} = F_M + F_A$, however, in the worst-case, none of the controllers failed, i.e., $f_{M,A} = 0$.

- Controller-To-Switch (C2S): After the consensus is reached for the corresponding request, in the case of NON-SELECTIVE mode all PRIMARY and SECONDARY controllers issue the reconfiguration responses to all affected switches ($E = 1$), while in the SELECTIVE mode, only PRIMARY controllers issue the responses ($E = 0$). If we consider that on average $|S_{cli}|$ switches are reconfigured based on the clients requests, the total number of responses issued by *controllers* towards *switches* is $|S_{cli}| \cdot (Req_P + E \cdot Req_S) \cdot m \cdot \mu_{cli}$. In the best-case, all faulty controllers were detected ($Req_P = 1$) and the working mode is SELECTIVE $E = 0$, thus only $|S_{cli}| \cdot m \cdot \mu_{cli}$ messages are needed. While in the worst-case, the initial case, $Req_P = F_M + 1$, $E = 1$, $Req_S = F_M + F_A + 1$.

Table 6.2: Exchanged number of messages on each communication channel.

Channel	MORPH	Best-Case MORPH	Worst-Case MORPH	No Fault-Tolerance	[128]
CLI2S	$m\mu_{cli}$	$m\mu_{cli}$	$m\mu_{cli}$	$m\mu_{cli}$	$m\mu_{cli}$
S2C	$(Req_P + Req_S)m\mu_{cli}$	$m\mu_{cli}$	$(2F_M + F_A + 1)m\mu_{cli}$	$m\mu_{cli}$	$(2(F_M + F_A) + 1)m\mu_{cli}$
C2C	$(\|C\| - f_{M,A} - 1)(Req_P + Req_S)m\mu_{cli}$	$(\|C\| - F_M - F_A - 1)m\mu_{cli}$	$(\|C\| - 1)(2F_M + F_A + 1)m\mu_{cli}$	-	-
C2S	$\|S_{cli}\|(Req_P + EReq_S)m\mu_{cli}$	$\|S_{cli}\|m\mu_{cli}$	$\|S_{cli}\|(2F_M + F_A + 1)m\mu_{cli}$	$\|S_{cli}\|m\mu_{cli}$	$\|S_{cli}\|(2(F_M + F_A) + 1)m\mu_{cli}$

The summary of the communication overhead is presented in Table 6.2, with corresponding notation summarized in 6.1. Table 6.2 also presents the required number of messages when *no fault-tolerance* is guaranteed, i.e., in the case of a single replica. We additionally present the communication overhead specific to the design presented in a related work [128]. It can be observed that MORPH requires the same or lower amount of messages on CLI2S, S2C, C2S channels depending on the current network state. Furthermore, as mentioned in Section 6.1, C2C communication is not discussed in [128], thus in contrast to MORPH, SDAs are not correctly handled there.

6.4 Evaluation and Results

We next present our evaluation methodology and showcase and discuss the observed MORPH's performance.

6.4.1 Evaluation Scenario

For our SDN application we have implemented a centralized resource-based path finding application, based on Dijkstra's algorithm [60]. We deploy an instance of the routing application in each controller replica. In the case of SIAs, the weight of each link in the observed topology is set to a constant delay value corresponding to the propagation delay. For the SDAs, we consider a load-balanced routing approach where the cost of each link depends on the current load.

To validate MORPH in a realistic environment, we have emulated the Internet2 Network Infrastructure Topology [1], as well as a standard Fat-Tree data-center topology. Both topologies were discussed in Section 5.4. We have enabled a configuration of MORPH to support up to $F_M = 5$ Byzantine and $F_A = 5$ Fail-Stop controller failures. For example, to allow for a BFT operation, we deploy $2F_M + F_A + 1 = 16$ controller replicas. We varied F_M and F_A individually from 1 to 5, so to allow for an evaluation of the overhead of added robustness against either failure type. The overhead of our design scales with the arbitrary number of tolerated failures F_M and F_A and is independent of the number of deployed switches. Each SDN controller implements the logic to execute the routing task as well as the `C-COMPARATOR` component that compares the C2C synchronization inputs and notifies the switches of new path configurations. An `S-COMPARATOR` agent is hosted inside each switch instance, and is enabled to listen for remote connections. All communication channels (ref. Fig. 6.1) are realized using Transmission Control Protocol (TCP) with enabled Nagle's algorithm [127].

The distributed control plane deploys an IBC channel. To realize the switching plane, a number of interconnected OVS virtual switches were instantiated and isolated in individual Docker containers.

[1] Internet2 Advanced Networking - https://www.internet2.edu/products-services/advanced-networking/

Similar to Section 5.4, for Internet2 topology we derive the link distances and inject corresponding propagation delays. The links of the Fat-Tree have the inherit processing and queuing delays. The arrival rates of the service embedding requests were modeled using a N.E.D. [89]. In the Fat-Tree, each leaf switch was connected to 2 client instances, bringing the total number of clients up to 16. Internet2 deployed one client per switch.

Controller placement: In Internet2, we leverage a controller placement that allows for a high robustness against the controller failures. We consider the exemplary placements introduced in [86]. Since the accompanying placement framework [2] was unable to solve the placement problem for a very high number of controllers (i.e., up to 16 in our case), we have executed the problem by placing up to 5 controllers multiple times for the same topology, targeting different optimization objectives (including response time, maximized coverage in the case of failures, load balancing etc.). We then ranked the unique nodes based on their placement preference and have selected the highest-ranked 16 nodes to host the MORPH controllers. The SDN controller replicas in the data-center topology were deployed on the leaf nodes, similar as in Section 5.4.

Table 6.3: Parameters used in evaluation of MORPH implementation.

Parameter	Intensity	Unit	Meaning
F_M	$[1, 2, 3, 4, 5]$	N/A	Max. no. of Byzantine failures
F_A	$[1, 2, 3, 4, 5]$	N/A	Max. no. of Fail-Stop failures
$\lvert C \rvert$	$[4, 7, 10, 13, 16]$	N/A	No. of deployed controllers
$1/\lambda_{F_M}$	$[5, 10, 15, 20]$	[s]	Byzantine failure rate
$1/\lambda_{F_A}$	$[5, 10, 15, 20]$	[s]	Fail-Stop failure rate
S	$[0, 1]$	N/A	SIA / SDA operation
E	$[0, 1]$	N/A	NON-SELECTIVE / SELECTIVE
T	Internet2, Fat-Tree	N/A	Topology type

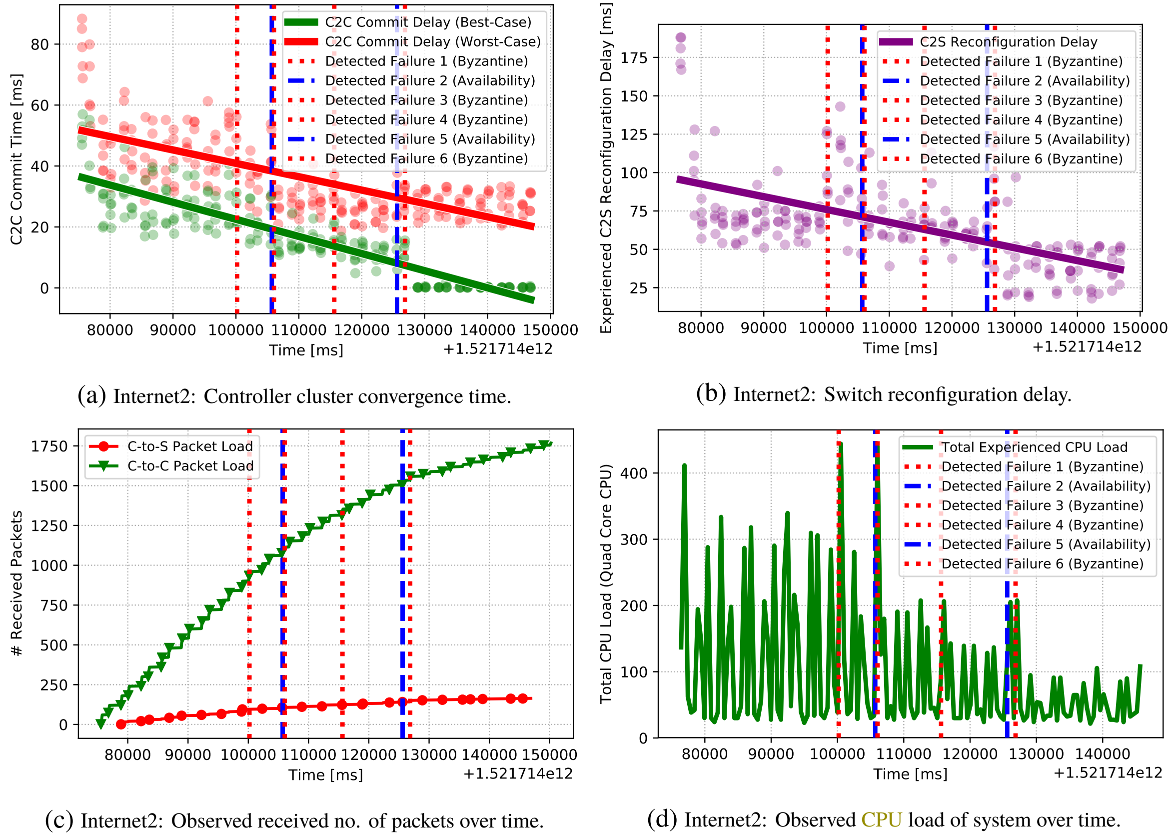
To evaluate the effect of the number, ratio and order of the Byzantine and Fail-Stop failures, we deploy a specialized fault injection component. The fault injection component generates the faults in an arbitrary active replica. The replicas are faulted with uniform probability. The ratio of injections is specified by bounding the maximum amount of individual failure types (ref. Table 6.3). The ordering of the different failure type injections is governed by the parametrization of mean arrival times as specified in Table 6.3. The intensity of failures follows the N.E.D. (similar to [3, 141]).

The OVS-based topology emulator, the `REASSIGNER` as well as up to *16* `C-COMPARATOR` and *34* `S-COMPARATOR` processes were deployed on a single commodity PC running a recent multi-core AMD Ryzen CPU and 32 GB of DDR4 RAM. We have used Gurobi Optimizer [3] to solve the ILP formulated in Section 6.3.2.

6.4.2 Overhead minimization by successful failure detection

Fig. 6.3 depicts the measured C2C and C2S delays, before, during and following the failure injection process. Similarly, the packet load and total system CPU utilization are depicted for the same measurement. Fig. 6.3 a) depicts the convergence time to the globally consistent state in the replicated

[2] Pareto Optimal Controller Placement (POCO) GitHub - https://github.com/lsinfo3/poco
[3] Gurobi Optimizer - http://www.gurobi.com/products/gurobi-optimizer



(a) Internet2: Controller cluster convergence time.

(b) Internet2: Switch reconfiguration delay.

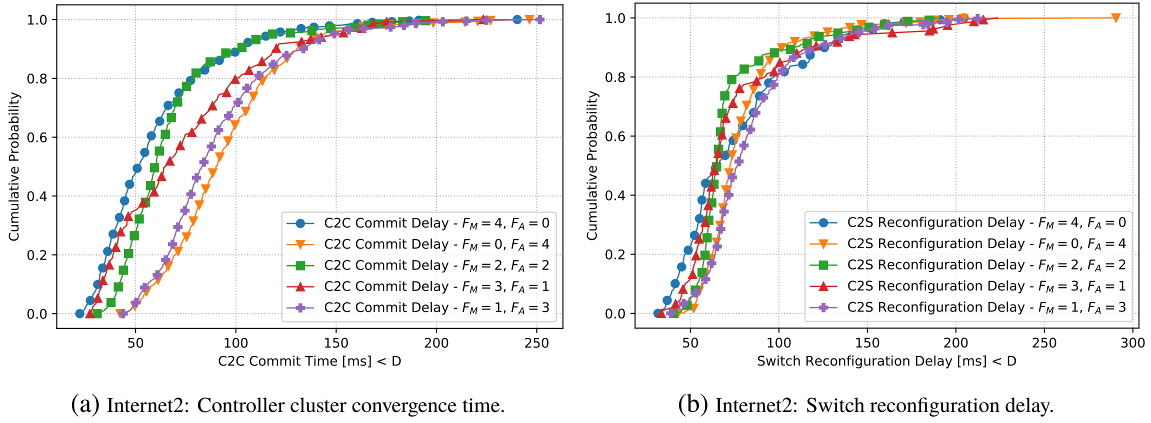
(c) Internet2: Observed received no. of packets over time.

(d) Internet2: Observed CPU load of system over time.

Figure 6.3: Successful detection of injected Byzantine and Fail-Stop failures leads to an instantaneous decrease in: a) the experienced C2C delay, both for the duration of the best- and worst-case state convergence time; b) the switch reconfiguration delay; c) the measured C2C and C2S packet arrivals (measured at the ingress of a correct controller and switch, respectively); and d) the total measured system CPU load. The depicted measurements are taken in the Internet2 topology, based on 13 controllers and 34 switches. The particular sample shown here depicts the case where 4 Byzantine and 2 Fail-Stop failures are identified in the depicted order (red and blue vertical dashed lines, respectively).

controller database. The *Best-Case* considers the time required to replicate and commit a state update in at least two controllers, while the *Worst-Case* considers the outcome where *every* controller converges to the globally consistent state. We observe that the initial failure injections consistently decrease the expected waiting time to achieve consensus and converge the state updates for the worst-case. The detection of later injections (Injection 5 and 6) has a lesser effect. For those detections, the trade-off in the added C2C delay overrides the benefits of the lower amount of controller confirmations required to update the controller state on each instance. This is a consequence of the displacement of the remaining active controllers in the Internet2 topology and hence the added C2C delay.

Fig. 6.3 b) depicts a drastic decrease in the response time required to handle client requests, i.e., to reconfigure each switch on the identified path with the new flow rules. We observe that *dynamically decreasing the minimum number of controllers needed to reach consensus on the correct configuration update, consistently lowers the expected time to reconfigure both the control and data plane*. Thus, MORPH increases the overall throughput of the SDN control plane, as additional service requests may be served in the same amount of time.



(a) Internet2: Controller cluster convergence time.

(b) Internet2: Switch reconfiguration delay.

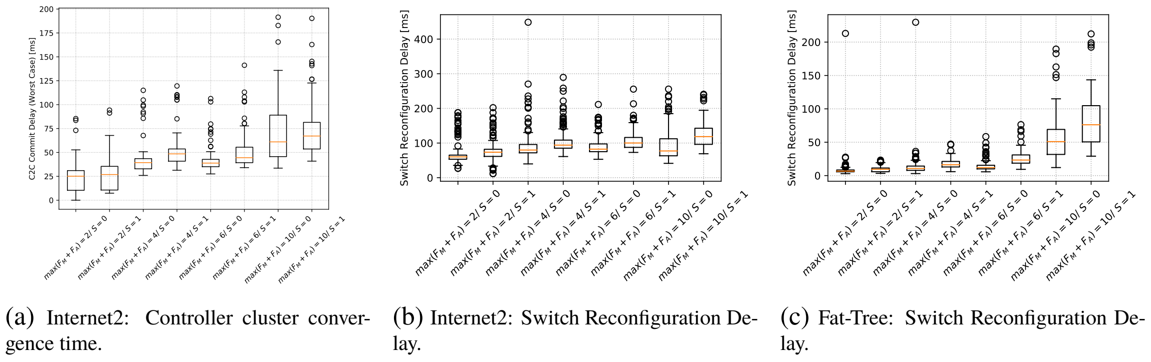
Figure 6.4: The type of the failure affects considerably the experienced C2C delay (depicted in Fig. 6.4 (a)). Indeed, the larger the number of detected Byzantine failures, the lower the instantaneous total number of PRIMARY controller messages required to process and commit new controller state update. Interestingly, the trend is observable for the switch configuration delay as well (depicted in Fig. 6.4 (b)), but not with the same intensity. The depicted figures result from the measurements taken for the Internet2 topology comprised of 13 controllers and 34 switches. The observation period includes the time preceding, during and following the successful failure injections.

Similarly, the exclusion of incorrect controllers from the message comparison procedure consistently lowers the control plane load. The minimization of C2C synchronization load can be observed from the change in the arrival slope in Fig. 6.3 c). The total CPU load is similarly decreased as both controllers and switches must process a lower total amount of controller messages with each replica exclusion. We have observed identical trends for the Fat-Tree.

Fig. 6.4 depicts the benefits of distinguishing the type of failure when reducing the number of active controllers used in the consensus procedure. According to Alg. 6, two message exchanges less are required to identify the correct message after detection of a Byzantine controller (F_M), while a single message less is required after discovery of an unavailable controller (F_A). Successful detection of Byzantine controllers thus leads to a decreased C2S and C2C reconfiguration time, compared to the discovery of the same amount of failures of a different type (or a combination of different faults). The detection of an unavailability-imposed failure (F_A) hence provides less information about the failed controller being benign or Byzantine. In that case, the `REASSIGNER` is more conservative when computing new C2S assignments.

6.4.3 Impact of SDN Application Statefulness

Fig. 6.5 depicts the consequence of the complexity of statefulness of the distributed SDN control plane on the experienced C2C and C2S delays. Stateful SDA applications necessitate consensus, which imposes an additional waiting time in the C2C synchronization. With consensus, the majority of controller configurations must match before deciding on the configuration message to be delivered to switch. After the consensus is reached, the controllers' data stores converge to a consistent state in each correct controller replica. The difference in the waiting time is not reflected in the data plane (switch) as much as in the C2C communication, since the switch experiences a constant waiting time for the



(a) Internet2: Controller cluster convergence time.

(b) Internet2: Switch Reconfiguration Delay.

(c) Fat-Tree: Switch Reconfiguration Delay.

Figure 6.5: Depending on an SDN application's requirement for access to the consistent controller data store, controllers impose a varying C2C synchronization overhead. Specifically in the case of resource-reservation applications, knowledge about the reserved and free resources is a requirement for the efficient client request processing (i.e., slicing, path embedding, load-balancing algorithms that rely on resource reservations). $S = 1$ denotes the case for a stateful controller application, while $S = 0$ denotes applications which do not demand causality in controller updates.

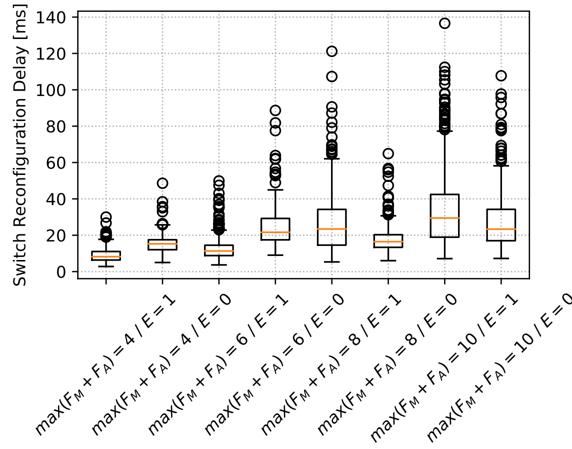


Figure 6.6: Difference in the observed switch reconfiguration delay for NON-SELECTIVE and SELECTIVE propagation of SECONDARY controller configurations in the [7..16] controller topology that tolerates $max(F_M + F_A) = [4..10]$ controller failures. Deployment of a large control plane induces a benefit in the average configuration time with the NON-SELECTIVE model. In a topology comprising a smaller number of controllers, the additional system workload related with the propagation of configurations on every SECONDARY controller negatively affects the overall performance.

minimum amount of required replicas Req_P, independent of the time it takes to reach consensus in the C2C communication.

However, as depicted both in Fig. 6.5 and Fig. 6.6, the overhead in the controller state and switch reconfiguration time is intensified by the *maximum number* of tolerated failures. The more failures the system is designed to tolerate, the higher the amount of required comparisons across the PRIMARY messages in order to forward the system state. We conclude that *the experienced controller and switch reconfiguration times scale with the number of tolerated controller failures*.

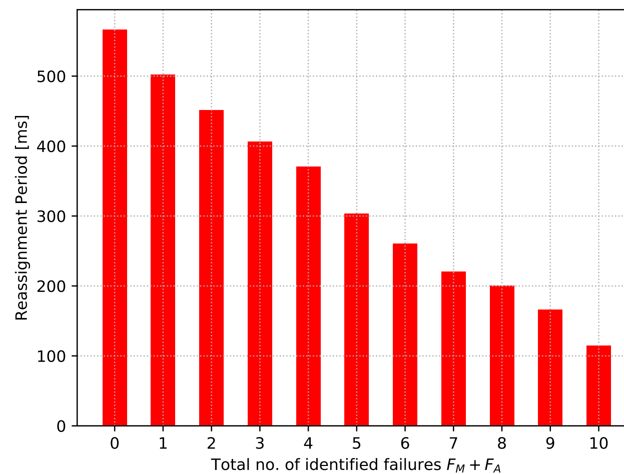


Figure 6.7: Decrease in the total execution time of the ILP solver for the C2S connection reassignment procedure in the Internet2 topology. The time required for the REASSIGNER to reconfigure the control plane scales inversely with the number of successfully identified controller faults. As the REASSIGNER tasks may execute asynchronously, without negatively disturbing the system correctness or control plane availability, we consider the deployment of MORPH in critical operation networks for practically feasible.

6.4.4 Impact of proactive propagation of switch configurations

Fig. 6.6 depicts the measurements taken in the Fat-Tree topology comprising 7 to 16 SDN controllers and 20 switches. A trend reversal in the switch reconfiguration delay can be observed, where for small-sized controller clusters (i.e., [4..10] controllers), the usage of the SELECTIVE mode ($E = 1$) results in a lower overall configuration delay. For large-scale control planes ([13..16] controllers), the trend is reversed and the NON-SELECTIVE mode ($E = 0$) becomes faster. We assume that the trend reversal is linked to the fact that in the average case where no failures are detected, SELECTIVE mode is more efficient than the NON-SELECTIVE one, since no additional packet overhead / CPU load is generated during the message processing, both in the control and data plane. In the large-scale control plane, the controller placement has a dominant role, so that the decrease in the C2S distance dominates the additional load related to the higher number of controller instances. For client requests where some controllers send incorrect configurations to the switches, or fail-silently and thus do not deliver any new configuration messages, the switch is able to deduce the correct majority only after experiencing an additional round-trip to collect additional configuration responses from the SECONDARY controllers. The NON-SELECTIVE model is intuitively faster in its worst-case, as it propagates every controller response directly to the switch after the computation of the configuration response, independent of the controller's role as PRIMARY or SECONDARY. Thus, no additional S2C round-trip delays are experienced when the received messages at the switch are inconsistent. Similarly, the switch is capable of collecting the Req_P consistent messages faster on average, since SECONDARY controllers may potentially deliver new configurations faster than the assigned PRIMARY instances.

6.4.5 Controller-Switch Reassignment Time

The ILP solver is executed once after the initial system deployment (to compute the initial S2C assignments) and once after each controller failure detection. Fig. 6.7 depicts a constant decrease in the execution time of the ILP as new faults are injected and failures are observed at the `REASSIGNER`. Indeed, the exclusion of incorrect controllers from the parameter space decreases the total running time of the ILP solver. Nevertheless, even in a large-scale formulation that assumes the assignment of 16 controller replicas to 34 switches, the `REASSIGNER` requires only up to $\sim 540ms$ to compute the optimal assignment w.r.t. delay and bandwidth constraints. Hence, MORPH supports the real-time switch reassignment and is thus deployable in online operation as well. For higher-scaled networks we do not foresee any limitations in the reassignment procedure related to the viability of our solution. Namely, when an inconsistency in the PRIMARY configuration messages is detected in the `S-COMPARATOR`, the switch initially ignores the inconsistent messages and continues to autonomously contact the SECONDARY replicas to request additional confirmation messages. Thus, the correctness of the system is never endangered and the `REASSIGNER` may proceed with its reassignment optimization procedure asynchronously in the background. MORPH is hence suitable for operation in critical networks where minimal downtime is a necessary prerequisite (i.e., in industrial control networks).

6.5 Chapter Summary

MORPH allows for distinguishing Byzantine from Fail-Stop controller failures at runtime, as well as for subsequent dynamic optimization of cluster membership.

Following a Byzantine or Fail-Stop controller failure, MORPH autonomously adapts the system configuration to minimize the distributed control plane overhead. By experimental validation, we have shown that MORPH achieves a performance improvement in both average- and worst-case system response time by minimizing the amount of required replicas when deducing new configurations. Thus, the worst-case waiting periods required to confirm new controller state updates and switch configurations are substantially reduced over time.

Furthermore, by excluding faulty controllers, the average packet and CPU loads incurred by the generation and transfer of controller messages to the switches are reduced over time. We have shown that the ILP formulation for QoS-constrained reassignment of C2S relationships may execute online, for both medium- and large-scale control planes comprising up to 16 controller replicas. The time required to adapt the system configuration scales proportionally with the number of successfully detected faulted controllers. Apart from the minimal additional CPU load related to the dynamic reassignment of controller connections, average- and worst-case computational and communication overheads are lower than those of comparable BFT SDN designs.

Enabling BFT generally results in a relatively high response time and communication load footprint in the normal case, i.e., where no faults are exposed. In the critical infrastructure networks, the trade-off of correctness and performance loss may, however, be acceptable [179, 227]. Nevertheless, the following Chapter 7 discusses the optimizations introduced to MORPH [1] in our subsequent

publications [6, 7, 11], that minimize the overhead of initial sequence ordering delay and control plane load.

Chapter 7

Lowering Response Time and Control Plane Overhead of the BFT Control Plane

Chapter 6 has discussed the importance of deploying BFT designs for achieving consensus on outputs of distributed controllers where a subset of replicas is Byzantine faulty. Realizing a BFT control plane, however, comes at a cost of additional replicas and increased response time and communication overhead penalties.

In the first half of this chapter, we make a point how an optimal separation of the controller cluster into *sufficiently-sized* Agreement and Execution (A&E) groups can lead to higher throughput and decreased control plane response times, compared to single-domain operation. Assuming heterogeneous resource availability, in the proposed approach, faster replicas are leveraged in the intersection of different A&E groups, while slower replicas run at their assigned speed without negatively influencing the faster replicas. We additionally introduce novel sequencing protocols to optimize the agreement step required for ordering of request updates in the BFT environment.

In the second part of the chapter, we investigate the benefits of offloading the procedure of controller outputs comparison, required for correct BFT operation, to carefully selected network switches. By minimizing the distance between the processing nodes and controller clusters / individual controller instances, we decrease the network load imposed by BFT operation. We coin the proposed extensions to MORPH as P4-Enabled BFT Control Plane (P4BFT). P4BFT's P4-enabled pipeline [46] is in charge of controller packet collection, correct packet identification and its forwarding to the destination nodes, thus minimizing accesses to the switches' software control plane and effectively outperforming the existing software-based solutions.

The designs presented henceforth were successfully validated and evaluated in small- and large-scale SDN control planes. *The proposed optimizations to MORPH design and their analysis were published and demonstrated in* [6, 7, 11].

7.1 Control Plane Partitioning into Multiple Agreement Groups

In order to support stateful controller-based applications (i.e., resource-constrained routing, stateful SDN-LB etc.), the replicas synchronize on the order of the state updates. Traditional BFT designs [53, 115], as well as MORPH, require active participation of all replicas in the administrative domain. They leverage an RSM approach to handle the client requests, where a specific majority of controller instances must come to the agreement about the order of the client requests, prior to executing these. Namely, distributed replicas reach consensus on the output of the computation in order to ensure the causality of decisions. We identify two associated overheads:

- **Execution overhead**: In existing BFT designs [53, 115], in order to preserve causality, non-faulty replicas always participate in all system operations. In the absence of faults, more replicas execute the decision-making requests than required to make progress, thus strongly limiting the execution throughput of the system. In environments where resources assigned to replicas vary (e.g., heterogeneous field-, edge-, cloud-devices) and / or can easily be scaled (e.g., using system virtualization), this leads to an under-utilization of fast replicas.
- **Agreement overhead**: Similarly, in existing BFT implementations [53, 115], replicas must reach a successful agreement on the order of execution of incoming client requests, or reach consensus on the update (as with MORPH, ref Sec. 6.2.6), prior to committing the resulting state updates. The agreement phase hence increases the total processing time. In this paper, we make a claim that the serialization of requests is a *mean to an end* and that the causality of configurations on individual external devices (i.e., switches) is a sufficient constraint to ensure the consistency of views among correct controller replicas.

Minimizing the execution overhead: In the approach proposed here, we optimally separate the controller cluster into *sufficiently-sized* A&E groups, thus enabling higher throughput and decreased control plane response times. Assuming heterogeneous resource availability, faster replicas are leveraged at the intersections of A&E groups, while slower replicas continue to run at their assigned speed without causing negative impact on faster replicas (in the average case). To identify A&E groups, we extend the ILP formulation for C2S assignment procedure presented in Section 6.3.2. The solver identifies the A&E groups for each deployed switch element at runtime, while maximizing the overlap of the members of different groups. The formulation considers the execution capacity of individual controllers, as well as the S2C delays as its constraints. The solver continues to execute at runtime, thus optimizing the assignment upon each discovered Byzantine orx Fail-Stop failure.

Minimizing the agreement overhead: We adopt the classical Practical BFT (PBFT) approach [53] to realize a distributed sequencer in order to minimize the fail-over time in the case of a leader failure. We additionally introduce a group-based variant of this protocol, that leverages the partitioning of the total controller set into multiple A&E groups to decrease the response time. In addition to two *agreement-based* designs, we introduce an *opportunistic* sequencing protocol design. With the opportunistic approach, successful handling of a client request implies reaching a consensus on a *consistent* device reconfiguration while preserving the *causality* of decisions, subsequent to the actual

request handling. We achieve the causality and agreement by reaching consensus: i) on the controller state at the time of application execution; ii) on the actual computed output result (to guarantee the consistency of decisions).

We have analyzed the overheads of switch reconfiguration time, the communication overhead and the request acceptances rates for each proposed sequencing protocol. We have executed the evaluation for emulated OVS-based Internet2 and Fat-Tree topologies with varied number of tolerated Byzantine failures.

7.1.1 Related Work - On Sequence and Value Agreement in BFT Designs

Agreement-based approaches have focused on the optimization of sequencing procedure by minimizing the number of replicas that actively participate in sequence proposals [61, 116]. *REBFT* [61] keeps only a subset ($2F + 1$ of a total of $3F + 1$) replicas active during normal case operation. It activates the passive replicas only after a detected replica fault. Such approaches rely on a trusted counter implementation to prevent *equivocation*, the capability of a Byzantine replicas to send conflicting proposals to other members. Since we do not assume a centralized proposer, we prevent equivocation by deciding new sequence numbers individually, without the overhead of a trusted counter nor passive replica activation delay.

Opportunistic BFT protocols have been investigated in [105, 128]. However, these approaches conclude about the consensus of the computed decisions based on the comparison of the instantaneous outputs and assume a *stateless* operation. In the contrast, in Opportunistic A Posteriori BFT (OBFT) we leverage the agreement procedure that relies on external outputs, i.e., *stateful* per-switch configurations that are inherent to network management scenarios.

Omada [65] is a sequencing-based BFT design that assigns replicas with either agreement or execution roles and parallelizes the agreement phase. It highlights the benefit of selecting a configuration with the lowest number of agreement groups. Contrary to our work, the authors assume a centralized sequencer per agreement group. Distinguishing causality property per configuration target is not discussed nor leveraged in their protocol. Similarly, Omada does not provide an insight into opportunistic approaches to execution handling.

7.1.2 BFT State Synchronization Protocols

In Section 6.2.5, we distinguished SIA from SDA SDN applications. In the remainder of the chapter, we solely consider the global SDA operations where client requests result in stateful write operations to the replicated data store. The subsequent client request executions must consider the preceding writes for their correctness. The value of the write operation is determined by an execution of a multi-phase BFT protocol. We henceforth distinguish *accepting* and *rejecting* BFT protocol executions. *Rejecting* executions are caused by replicas that interrupt the run due to a missing consensus in one of the protocol *phases* (e.g., caused by conflicting sequence number proposals, inconsistent execution outputs or packet loss). We assume that clients re-transmit the requests until a successful execution has been acknowledged by the controllers.

Distribution of state updates assumes an *eventually synchronous* model [126], where different replicas possess different views of the current configuration state for a limited time duration. Eventually, given an appropriately long quiescent period, all correct replicas converge to the same state. We assume that a bounded number of controllers may exhibit Byzantine behavior and / or Fail-Stop failures, respectively.

The proposed sequencing protocols guarantee the following properties:

- *Uniform Agreement*: When a correct replica commits a particular internal state / switch update (i.e., computes a particular response), all correct replicas eventually commit the same update.
- *Liveness*: All correct replicas eventually finalize the processing of each client request. The resulting run is declared *accepting* or *rejecting*.
- *Causality*: The updates to the controller data store and the per-switch configuration updates are executed in a causally dependent order. The controller's decision to reconfigure a switch take into account all preceding configurations of that switch.

We assume an assignment of a total of $2F_M + F_A + 1$ controllers per A&E group in order to tolerate an upper bound of individual F_M Byzantine and F_A Fail-Stop controller failures in that particular A&E group. We next introduce the three BFT protocols: the agreement-based Modified PBFT (MPBFT) and Serialized A Priori BFT (SBFT) protocols, and the opportunistic OBFT (ref. Table 7.1), used in the sequence number generation on each new client request.

Table 7.1: Overview of presented BFT protocols.

Algorithm	Name	Type	Number of Rounds
MPBFT	Modified PBFT	Agreement-based	2
SBFT	Serialized A Priori BFT	Agreement-based	3
OBFT	Opportunistic A Posteriori BFT	Opportunistic	2

7.1.2.1 Pre-Serialization Model MPBFT (Agreement-based)

MPBFT imposes exactly one A&E group for the whole administrative domain, with each active replica tasked with sequencing task and the execution of an agreed command. The workflow of MPBFT is visualized in Fig. 7.1. A *request-initiating* client initially invokes its application request to all active replicas (`REQUEST` phase). For each incoming client request, each replica assigns a unique sequence number and distributes the proposed sequence number to other replicas (`PREPARE` phase). The replicas compare the sequence number proposals. If the correct majority of proposals are matching (i.e., the same sequence number is proposed by the majority of correct replicas), successful global agreement has been reached. At the begin of the `COMMIT` phase, each correct replicas executes the client request. The execution output is subsequently broadcasted by each replica to the remainder of the cluster and the collected output responses are once again compared in all replicas. Each controller replica deduces the correct majority response and eventually commits the output to its local data store (i.e., a store of reservations) and finally reports the agreed output to the target clients (`REPLY` phase). After collecting $F_M + 1$ consistent output messages, the *target clients* (e.g., switches) apply the new configuration.

MPBFT is a variation of PBFT [53] that requires no leader and is thus tolerant to individual node failures. Compared to PBFT, we shorten the protocol execution by one round. Whereas PBFT proposes a `PRE-PREPARE` round, MPBFT skips this round by leveraging a client-initiated atomic multicast execution and a *distributed sequencer*. Namely, each new client request is multicasted to each replica of the system. The replicas propose a new sequence number for the request by incrementing the current counter as per Alg. 10. The sequence numbers for new client requests are assigned based on the current state of a local atomic counter. Following an arrival of a new request, the replicas yield the lowest unallocated sequence number and propose this value to the remaining replicas. After collecting *sufficient* matching `PREPARE` messages, all *correct* replicas decide to accept the sequence number contained in the correct majority proposal as the final sequence number for this request. Table 7.3 summarizes the exact amounts of required matching messages to progress the protocol execution.

If correct majority vote is not achieved during the agreement process on either the sequence number or the computed output, the replicas respond with a rejection. If sufficient rejection messages are collected, the current execution is canceled and the run is declared *rejecting*. Concurrent client requests can lead to assignment of equal sequence numbers to different requests at different replicas, thus resulting in rejecting runs.

The execution capacity of MPBFT is limited by the slowest replica in the system. Consider the scenario $F_M = 1, F_A = 0$ depicted in Fig. 7.1. Each controller C_i is able to service request workload up to a capacity of P_i per observation interval. The portrayed system is thus able to service computations up to $max(\sum_{i=1..N} \lambda_i) \leq min(P_i)$, or 500 requests / interval (imposed by the capacity of $C4$ and $C5$). Thus, Client 1 (with processing requirement of $\lambda_1 = 500$) and Client 2 ($\lambda_2 = 400$) cannot be serviced concurrently. One can alternatively portray the depicted rates as continuous execution workloads. While active participation of $C4$ and $C5$ in the system is unnecessary to tolerate a single Byzantine fault, they are included in sequencing and execution steps and are necessary to progress the system state. MPBFT's communication overhead is quadratic (ref. Table 7.4).

With alternative protocol designs SBFT and OBFT, we next leverage the additional execution capacity by partitioning the control plane into multiple A&E groups.

7.1.2.2 Pre-Serialization Model SBFT (Agreement-based)

With SBFT, agreement and execution processes are administered by multiple A&E groups. We assign for each request-initiating client (e.g., an NBI application or an end-point) an A&E group according to the algorithm in Section 7.1.3. To tolerate F_M Byzantine and F_A Fail-Stop failures in the scope of a single A&E group, each group must comprise $2F_M + F_A + 1$ replicas. Multiple execution groups can process the client requests concurrently.

SBFT's design is depicted in Fig. 7.2. Compared to MPBFT, SBFT introduces the `PRE-PREPARE` step, where replicas belonging to the A&E group propose and subsequently notify the remaining replicas of the assigned sequence number. In an *accepting* run, the group replicas collect the responses in the `PREPARE` phase and reach consensus by collecting $\lceil \frac{|C|+F_M+1}{2} \rceil$ matching sequence number proposals. Finally, the replicas of the A&E group execute the request in the `COMMIT` phase and broadcast the

Algorithm 10 Logical Sequencer: Ordering of client requests.

Notation:
M_P Client request (e.g., flow request)
M_C Replica message (sequence number proposal) initiated at a remote controller
C Set of available SDN controller replicas
R_{ID} Unique client request identifier
$\mathcal{R}_{mappings}$ Mapping of client request identifiers to unique sequence numbers
S_{atomic} Atomic sequencer that yields the current sequence number

```
1:  upon event on-client-request < M_P, R_ID > do
2:  ...
3:  proposed_seq_no = propose_seq_no(R_ID)
4:  ...
5:
6:  upon event on-new-replica-sync-update < M_C, R_ID > do
7:  ...
8:  switch PHASE do
9:      case MPBFT-PREPARE:
10:         propose_seq_no(R_ID)
11:     case SBFT-PRE-PREPARE:
12:         propose_seq_no(R_ID)
13: ...
14:
15: function PROPOSE_SEQ_NO(R_ID)
16:     if R_ID ∈ R_mappings then
17:         return R_mappings[R_ID]
18:     else
19:         while S_atomic ∈ R_mappings.values() do
20:             S_atomic = S_atomic + 1
21:         R_mappings[R_ID] = S_atomic
22:         return R_mappings[R_ID]
```

response to all remaining replicas. If $F_M + 1$ matching outputs are received, the replicas update the internal state and notify the target clients of the final result in `REPLY` phase. The communication overhead of SBFT is bounded $O(3|\mathcal{A}||C|)$, and grows linearly for a fixed A&E group size.

Causality: To ensure that the causality property holds in MPBFT and SBFT, the controllers execute the sequenced request in order agreed during `PREPARE`. The replicas execute the `COMMIT` phase only if the outputs (i.e., the added reservations) for the preceding requests were seen by the executing replica. Thus, prior to handling subsequent requests, the status of each preceding run (*accepting* / *rejecting*) must be determined first.

7.1.2.3 Post-Negotiation Model OBFT (Opportunistic)

OBFT is a speculative take on SBFT, where request executions run prior to reaching consensus on the computed output values. A global sequencer is not used in OBFT and thus `PRE-PREPARE` and `PREPARE` phases are omitted. Instead, each replica maintains the hashes of current switch configurations, as well as a state array containing the hashes of the configurations of the switches at the time of request executions (Time Of Request (TOR) hashes). Following the output computation in the `COMMIT` phase, the replicas come to consensus about the updated switch state in the `PRE-REPLY` phase. This workflow is depicted in Fig. 7.3.

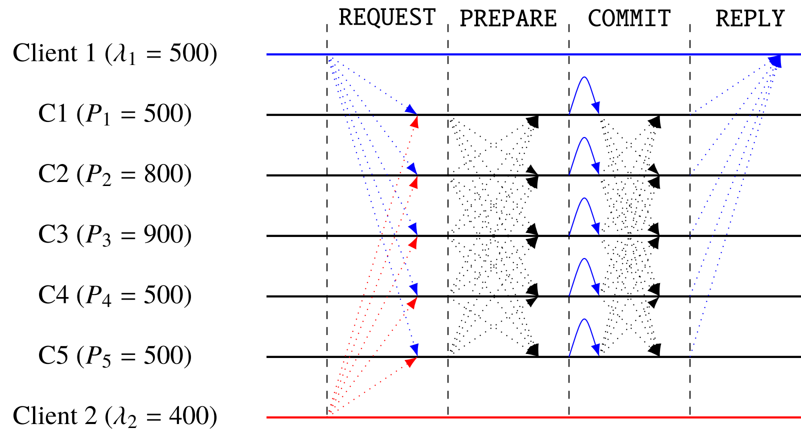


Figure 7.1: MPBFT: In REQUEST phase, the clients initiate new executions. During PREPARE, controller replicas agree on the execution order by reaching consensus on the assigned sequence number for the clients' requests. Each replica executes the request in the COMMIT phase. During REPLY, target clients are notified of reconfigurations. Client 2's requests cannot be serviced as a result of a limited processing capacity of the control plane.

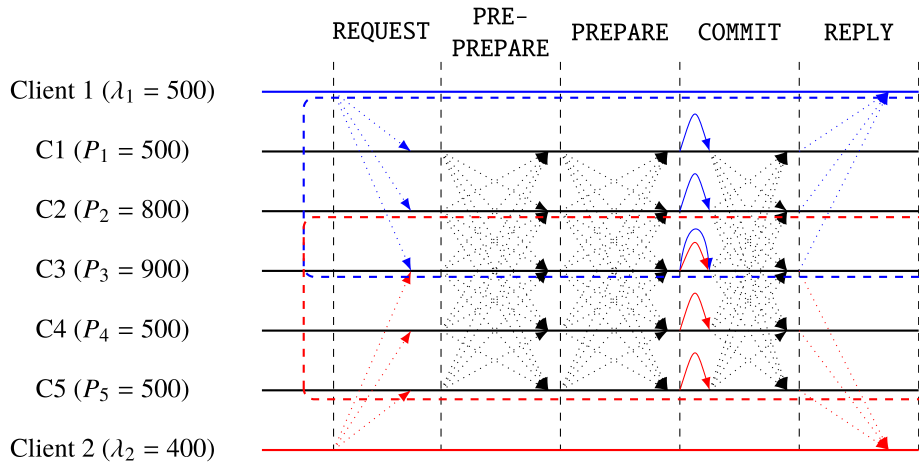


Figure 7.2: SBFT: Compared to MPBFT, SBFT allows for more efficient allocation of execution resources, since execution is separated into multiple A&E groups. The separation, however, comes with an overhead of a PRE-PREPARE step, where replicas of a serving A&E initiate a global agreement by transmitting their proposed sequence number to all other replicas. Thus, potential conflicts across different A&E groups are detected in the PREPARE, prior to COMMIT step.

In contrast to MPBFT and SBFT, in their COMMIT phase, the replicas of an A&E group compute the outputs, and in addition to the computed response outputs, they individually broadcast the hash arrays denoting their view of the target clients' configurations. *If and only if* the following three constraints are met, the run is *accepting*:

- Acceptance Constraint 1 (COMMIT): Each accepting replica *that is not part of the serving A&E group* evaluates its actual current local view of the switch states against the individual configuration proposals, and: i) *iff* $F_M + 1$ matching output values have been computed by the

executing A&E replicas; and b) the current local view of switch configuration hashes is matching with that of the individual executing A&E replicas; at beginning of `PRE-REPLY` they respond to the executing replica with an *accepting* status. Otherwise, they respond with an *rejecting* status.

- Acceptance Constraint 2 (`COMMIT`): Similarly, the executing replicas of the A&E group compare the proposed hash array with their local TOR hashes for individual target clients (i.e., target switches) and respond to all A&E replicas of acceptance or rejection, at the beginning of `PRE-REPLY` .

- Acceptance Constraint 3 (`PRE-REPLY`): If *sufficient* (ref. Table 7.3) positive confirmations have been collected at the end of `PRE-REPLY` phase, each replica internally commits the output proposed by the correct majority of the A&E group. The A&E group replicas then notify the configuration targets of the agreed output (`REPLY` phase).

OBFT's communication overhead is quadratic and grows with $|C|$.

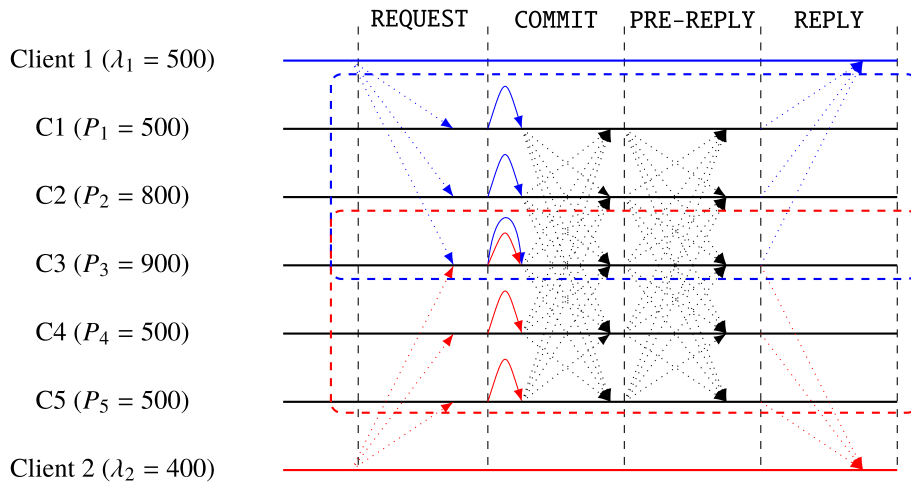


Figure 7.3: OBFT: An opportunistic BFT protocol variation, where A&E group members execute their clients' requests prior to the distribution of reference state configurations based on which the computations were executed. The internal controller state and the target clients are updated only if the consensus on the reference state configurations could be reached for the *correct* majority of global controller instances (ref. Table 7.3).

7.1.3 Enhancing MORPH's Controller-to-Switch Assignment Model

In our design, each request-initiating client (e.g., an NBI client or a switch itself) is assigned either an existing or a new unique A&E group (if a particular A&E combination of replicas does not exist yet). The replicas of different A&E groups are allowed to overlap.

The following extended C2S assignment dictates the per-switch A&E groups (i.e., sets of replicas), comprising replicas allowed and required of contacting target switches when reconfigurations must be applied as the output of client requests. The following ILP formulation of the assignment problem maximizes the total overlap between the members of all active A&E groups, so to minimize the

Algorithm 11 Hash comparison in the OBFT-COMMIT phase.

Notation:
R_{ID} Unique client request identifier
$\mathcal{HV}$ Current configuration hashes for the switches
$\mathcal{HVC}[\mathcal{R_{ID}}]$ Switches' config. hashes prior request computation (TOR)
find-path() An exemplary SDN application logic operation
consensus() Returns consensus message according to the number of minimum required confirmations (ref. Table 7.3)
$m^{C}_{R_{ID}}$ A `COMMIT` message for round R_{ID}
$M^{C}_{R_{ID}}$ Set of buffered `COMMIT` messages for R_{ID}

1: **procedure** HANDLE NEW CLIENT REQUEST
2: **upon event** *on-received-client-request* $(CL_{R_{ID}})$ **do**
3: $\mathcal{R}$ = *find-path*($CL_{R_{ID}}$.routing_request)
4: **for all** SW $\in \mathcal{R}$ **do**
5: current-hash[SW] = *hash*(SW.state)
6: $m^{C}_{R_{ID}}$.hash, $m^{C}_{R_{ID}}$.path = current-hash, $\mathcal{R}$
7: *broadcast-to-cluster-members*($m^{C}_{R_{ID}}$)
8:
9: **procedure** HANDLE INCOMING COMMIT MESSAGE
10: **upon event** *new-replica-sync-message*($m^{C}_{R_{ID}}$) **do**
11: $P^{R_{ID}}_{C}$ = *consensus*($M^{C}_{R_{ID}}$, <val, state-hash-array>)
12: *on-init-obft-pre-reply*($P^{R_{ID}}_{C}$,inline-with-replica-view($P^{R_{ID}}_{C}$))
13:
14: **function** INLINE-WITH-REPLICA-VIEW($P^{R_{ID}}_{C}$)
15: **for all** SW $\in P^{R_{ID}}_{C}$.path **do**
16: **if** $\mathcal{HV}[\text{SW}] == P^{R_{ID}}_{C}.hash[\text{SW}]$ **then**
17: *pass ()*
18: **else if** $\mathcal{HVC}[R_{ID}][\text{SW}] == P^{R_{ID}}_{C}.hash[\text{SW}]$ **then**
19: *pass ()*
20: **else** *return* (REJECT)
21: *return* (ACCEPT)

Table 7.2: Notation used in Tables 7.3, 7.4 and 7.5.

Symbol	Meaning
C	Set of active controllers in the system
F_M	Number of tolerated Byzantine failures in a single A&E group
$Req(t)$	Time-variant number of replicas [1] that must be assigned to each switch
$\mathcal{S}$	Set of switches in the system
P_{C_i}	Total available controller C_i's capacity.
L_{CL_k}, L_{S_j}	Request processing load stemming from the NBI client CL_k and edge switch S_j, respectively.
$D_{C,S}$	Maximum tolerable delay for C2S communication.
$\mathcal{A}$	Controller replicas belonging to a single A&E group
$\lvert\mathcal{M}_{agr}\rvert$	Sum of the tolerated Byzantine failures and the majority of correct replicas per A&E group: $\lceil\frac{\lvert\mathcal{A}\rvert+1+F_M}{2}\rceil$
$\lvert\mathcal{M}_{glob}\rvert$	Sum of the tolerated Byzantine replicas and the majority of all correct active replicas: $\lceil\frac{\lvert C\rvert+1+F_M}{2}\rceil$
`CMP`	Computation overhead of executing the packet comparison
`E`	Computation overhead of executing SDN application

synchronization delay during the consensus steps of the presented BFT protocol. The proposed re-assignment mechanism, the new objective function and the constraints extend the MORPH's formulation [1]. We do not repeat descriptions of each constraint in detail here, but refer the reader to the summary in Table 7.5. The optimization is executed at system startup and dynamically at runtime, on each detected controller failure or new client addition (incl. new switches after topology extensions).

Table 7.3: Number of matching messages required to reach consensus in the respective protocol phase (worst-case).

Algorithm	PRE-PREPARE	PREPARE	COMMIT	PRE-REPLY	REPLY
MPBFT	N/A	$\|M_{glob}\|$	$F_M + 1$	N/A	$F_M + 1$
SBFT	$\|M_{agr}\|$	$\|M_{glob}\|$	$F_M + 1$	N/A	$F_M + 1$
OBFT	N/A	N/A	$\|M_{agr}\|$	$\|M_{glob}\|$	$F_M + 1$

Table 7.4: Computational and communication overhead of the introduced BFT protocols.

Algorithm	Computational Overhead	Communication Overhead
MPBFT	$O(2\|C\|\texttt{CMP} + \|C\|\text{E})$	$O(2\|C\|\|C\|)$
SBFT	$O(\texttt{CMP}(2\|C\| + \|\mathcal{A}\|) + \|\mathcal{A}\|E)$	$O(3\|\mathcal{A}\|\|C\|)$
OBFT	$O(2\|C\|\texttt{CMP} + \|\mathcal{A}\|E)$	$O(\|\mathcal{A}\|(\|C\| + 1) + \|C\|(\|C\| - 1))$

For each switch S_i we can derive a bitstring R_{S_i} comprised of *ones* for replicas actively assigned to S_i and *zeros* for the unassigned replicas. We then formalize the objective function:

$$min \sum_{S_j \in \mathcal{S}} \sum_{S_i \in \mathcal{S}, S_i \neq S_j} HD(R_{S_j}, R_{S_i}) \tag{7.1}$$

where $HD(R_{S_j}, R_{S_i})$ denotes the Hamming distance between the assignment bit-strings for S_j and S_i. Combined with the adapted *minimum assignment* constraint depicted in Table 7.5, we ensure the building of minimum-sized A&E groups that fulfill the capacity and delay constraints of the clients.

Note: A wary reader may notice that, for simplification, we do not distinguish between MORPH's PRIMARY and SECONDARY replicas in Table 7.5. Note, however, that with the extended objective presented here, we remain compatible with all optimizations introduced in Section 6.3.

Table 7.5: Constraints used in building the A&E groups.

Constraint	Formulation
Minimum Assignment	$\sum_{C_i \in C} A_{C_i,S_j} == Req(t), \forall S_j \in \mathcal{S}$
Unique Assignment	$A_{C_i,S_j} \leq 1, \forall C_i \in C, S_j \in \mathcal{S}$
Bounded Capacity	$\sum_{S_j \in \mathcal{S}} A_{C_i,S_j} \cdot L_{S_j} \leq P_{C_i} - \sum_{CL_k \in C\mathcal{L}} L_{CL_k}, \forall C_i \in C$
Delay Bounds	$A_{C_i,S_j} \cdot d_{C_i,S_j} \leq D_{C,S}, \forall C_i \in C, S_j \in \mathcal{S}$

7.1.4 Evaluation and Results

To evaluate the different BFT protocols, we realized a centralized path computation application that executes in each of the SDN replicas. The routing algorithm leverages Constrained Shortest Path Forwarding (CSPF) to choose the optimal path w.r.t. bandwidth resource consumption weight. The BFT protocol executions take the source-destination pair and the required bandwidth as an input for the service request. Subsequently, the BFT protocol execution executes the application logic (path computation) in the `COMMIT` phase and, in case of an accepting run, reconfigures the switches on the agreed path in the `REPLY` phase. On each successful embedding, the weights of the links on the computed path are increased by the flow's requested bandwidth.

To evaluate the designs of all three protocols, we consider the following performance metrics: i) time required to apply a new switch reconfiguration, measured from the time of a client request arrival

until the confirmation of the last switch reconfiguration; ii) the acceptance rate for the new arrivals; iii) the total communication overhead.

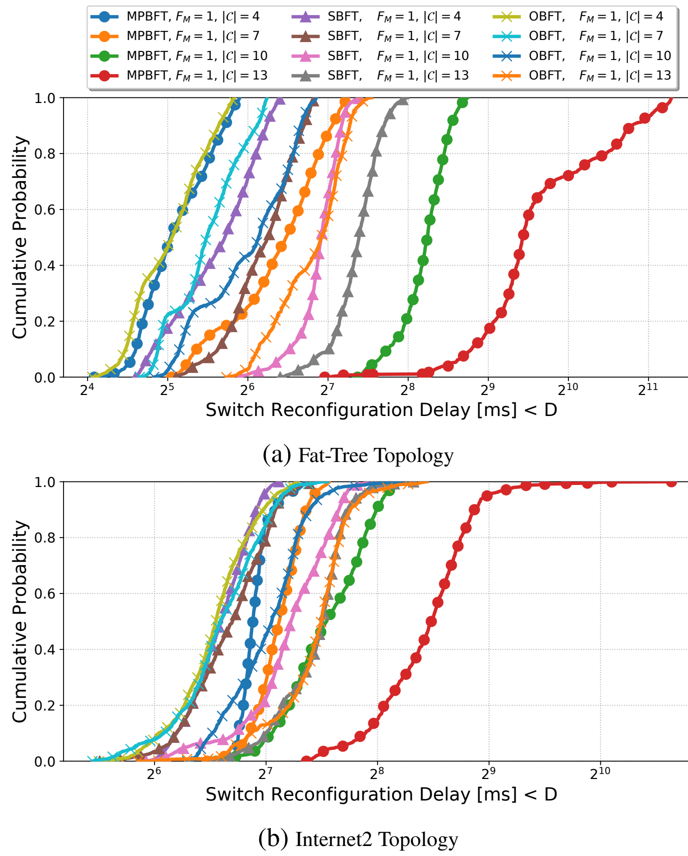
Once again, we validate our claims in Internet2 and Fat-Tree topologies, encompassing 34 and 20 switches, respectively. C2S and C2C communication is realized using the IBC channel. Parameter settings for network delay emulation are as described in Sections 5.4 and 6.4.1. A single client was placed at each switch of the Internet2 topology, while two clients were placed at each leaf-switch of the Fat-Tree topology. The arrivals for incoming service requests are modeled using N.E.D. [89].

To generate the hashes for per-switch configuration state (ref. Section 7.1.2.3), we used Python's *hashlib* implementation and the Secure Hash Algorithm (SHA-256), defined in FIPS 180-2 [220]. We used Gurobi to solve the ILP formulated in Section 7.1.3. The measurements were executed on a commodity PC equipped with AMD Ryzen 1600 CPU and 32 GB RAM.

7.1.4.1 Total Response Time for Reconfigurations of Internal Controller and Switch State

Fig. 7.4a and Fig. 7.4b depict the accumulated response time starting with the reception of a client request at a controller replica until modification of the last reconfigured switch on the computed path. The total number of active controllers was fixed to $|\mathcal{C}| = 10$ and the measurement was executed for A&E group sizes varying between $|\mathcal{A}| = 3$ and $|\mathcal{A}| = 7$ replicas (for $F_A = 0$ Fail-Stop, and $F_M = 1$ and $F_M = 3$ Byzantine failures, respectively). Rejecting executions were not considered. Both Fat-Tree and Internet2 topologies depict the benefit of opportunistic execution and a lower number of phases in OBFT in all scenarios.

In Fig. 7.4a and Fig. 7.4b, we vary the total number of deployed controllers. All three protocols provide for similar performance for controller constellations where the A&E group size approximately equals the total number of active replicas (i.e., all replicas in the same A&E group). After provisioning additional replicas (case for $|\mathcal{C}| = [7..13]$), MPBFT's performance decreases compared to SBFT and OBFT. This is due to MPBFT requiring interactions between all replicas for successful request handling, whereas SBFT and OBFT continue to operate at the level of a constant A&E group size. OBFT offers best performance in both topologies. This is due to SBFT and MPBFT requiring additional rounds to handle request sequencing, in contrast to OBFT, which ensures causality property holds even in the case of unordered executions. Additionally, MPBFT's performance suffers due to commands' execution on each controller replica. On average, MPBFT's agreement requires consideration of a larger number of replicas than SBFT and OBFT. Internet2 topology depicts a lower discrepancy between SBFT and OBFT and highlights the benefit of sequencing in geographically distributed scenarios where network delays cause a longer asynchronous period and thus a higher probability of execution overlaps (confirmed by Fig. 7.5). The maximum path lengths are higher for Internet2 topology, thus resulting in a higher number of overlapping reservations that cause execution rejections / stalling period in opportunistic OBFT.



(a) Fat-Tree Topology

(b) Internet2 Topology

Figure 7.4: The accumulated control plane response time for varied numbers of active controller replicas $|\mathcal{C}| = [4..13]$, $F_A = 0$ and an A&E group size of $|\mathcal{A}| = 3$. While OBFT portrays the lowest reconfiguration delays, its performance is similar to SBFT and MPBFT for small control planes, slightly better compared to SBFT and largely dominant compared to MPBFT for larger topology sizes.

7.1.4.2 Acceptance rates for arriving requests

In Fig. 7.5 we vary the per-client arrival rates λ for incoming client requests. With $\lambda = 4$, on average, 64 $\frac{requests}{s}$ are processed by the cluster in Internet2 topology. Opportunistic execution of OBFT and its late hash comparison tends to result more often in rejecting runs, compared to SBFT that serializes all requests prior to their processing. MPBFT results in a relatively high percentage of rejections, due to a higher chance of conflicting sequence number handouts that may occur concurrently since all replicas are involved in proposals during PREPARE phase.

7.1.4.3 Communication overhead

Fig. 7.6 depicts the relation between communication overhead and number of active controllers. C2S communication overhead increases with the number of replicas that execute the request and communicate their result to the target switches. Following a response computation in MPBFT, each replica distributes the newly computed configurations to switches, hence the linear overhead increase. Since the size of the A&E group remains unchanged throughout all depicted scenarios, SBFT and OBFT have a constant low C2S overhead. The C2C overhead scales with the number of active replicas in the A&E group. For MPBFT and OBFT, this increase is quadratic. For SBFT, the C2C overhead

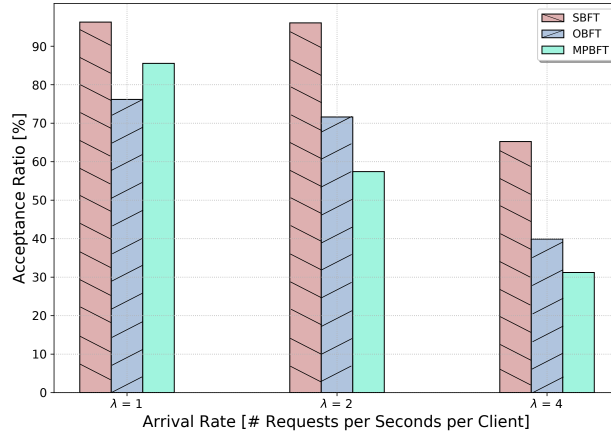


Figure 7.5: Acceptance rates for incoming client requests in Fat-Tree and $F_M = 1, |C| = 4$. SBFT tends to execute a higher number of *successful* runs compared to: i) MPBFT, due to its larger number of active replicas involved in sequencing process and ii) OBFT, due to its opportunistic design, where consistency of outputs is agreed upon, only after execution has finished.

increase is linear. It should be noted that the linear evolution holds only for constant A&E group sizes, i.e., for fixed F_M and F_A.

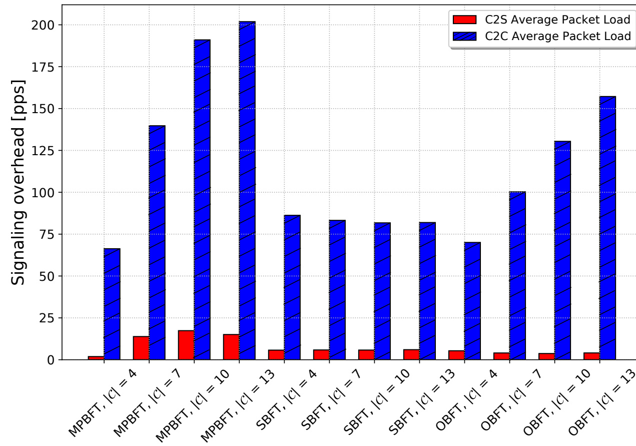


Figure 7.6: Signaling overhead $[pps]$ when serving 16 $\frac{requests}{s}$ for a varied number of controllers $|C| = [4..13]$ and a fixed A&E group size $|\mathcal{A}| = 3$. SBFT possesses the lowest overhead and linear growth, followed by OBFT and MPBFT, that have a quadratic growth scaling with $|C|$.

Additional notes: While SBFT and MPBFT ensure a single execution and validation of inputs for client requests (i.e., each client sequence number is mapped to a unique request), OBFT executes client requests speculatively, prior to reaching consensus. Thus, Byzantine clients may attempt affecting the order of execution, or generate execution contentions. Metering mechanisms for misbehaving clients and their exclusion could cater for this case. They are, however, not in the scope of this work.

7.2 Control Plane Acceleration by Offloading Tasks to the Data Plane

Apart from our publications [1] and [7], BFT has been investigated in the context of distributed SDN control plane in [115, 128]. The related state-of-the-art designs assume the deployment of SDN controllers as a set of RSMs, where clients submit inputs to the controllers, that process them in isolation and subsequently send the computed outputs to the target destination (i.e., reconfiguration messages to destination switches). They assume trusted platform execution and a mechanism in the destination switch, capable of comparison of controller messages and deduction of the correct message. Namely, after receiving $F_M + 1$ matching payloads, the observed message is regarded as correct and the containing configuration is applied.

The presented models are suboptimal in a few regards.

First, they assume the collection and processing of controller messages exclusively in the receiver nodes (configuration targets). Propagation of each controller message carries a large load footprint in large-scale IBC SDNs, due to replication of control channels for each replica. Second, they neglect the incurred performance overhead of message comparison procedure in the target switches. The existing BFT designs [1, 7, 115, 128] all realize the packet comparison procedure solely in software. The non-deterministic / varied latency imposed by the switch CPU (i.e., the slow path [77]) may, however, be limiting in use cases that place requirements on bounded control plane response time, such as for the industrial TSN or 5G scenarios [229]. This motivates a hardware-accelerated BFT design that minimizes or even bounds the processing delays.

In the remainder of the chapter, we investigate the benefits of offloading the procedure of comparison of controller outputs, required for the correct BFT operation, to carefully optimally selected network switches. By minimizing the distance between the processing nodes and controller clusters / individual replicas, we decrease the network load imposed by BFT operation. P4BFT's P4-enabled pipeline is in charge of controller packet collection, correct packet identification and its forwarding to the destination nodes, thus minimizing accesses to the switches' software control plane and effectively outperforming the software-based solutions.

7.2.1 Related Work - On Data Plane-Accelerated BFT

In the context of centralized network control, BFT is still a relatively novel area of research. Reference solutions [115, 128], assume the comparison of configuration messages, transmitted by the controller replicas, in the switch destined as the configuration target. With P4BFT, we investigate the flexibility advantages of message processing in any node capable of message collection and processing, thus allowing for a footprint minimization. [128] and [1] discuss the strategy for minimization of number of matching messages required to deduce correct controller decisions, which we adopt in P4BFT.

Recently, Dang et al. [57] have portrayed the benefits of offloading coordination services for reaching consensus to the data plane, on the example of a Paxos implementation in P4 language. With P4BFT, we investigate if a similar claim can be transferred to BFT algorithms in SDN context. In the same spirit, in [200], end-hosts partially offload the log replication and log commit operations of Raft

to neighboring P4 devices, thus accelerating the overall commit time. In the context of in-network computation, Sapio et al. [164] discuss the benefit of data aggregation offloading to constrained network devices for the purpose of data reduction and minimization of workers' computation time.

7.2.2 P4BFT: Enhancing the MORPH model

We coin the approach presented henceforth as P4BFT. P4BFT allows for collection of controllers' packets and their comparison in *processing* nodes, as well as for relaying of deduced correct packets to the destinations. It selects the optimal processing nodes at per-destination-switch granularity. The proposed objective minimizes the control plane load and reconfiguration time, while considering constraints related to the switches' processing capacity and the upper-bound reconfiguration delay. It executes in software, e.g., in P4 switch behavioral model (*bmv2*[1]), or in physical environments, e.g., in Netronome SmartNICs[2]. Correctness, processing time and deployment flexibility of P4BFT are validated in both environments.

Apart from system model assumptions presented in Section 6.2, we henceforth consider flexible function execution on the networking switches for the purpose of BFT system operation. The flexibility of in-network function execution is bounded by the limitation of the data plane programming interface (i.e., the $P4_{16}$ [46] language specification in the case of P4BFT). The C2S and C2C communication is realized using an IBC channel [169].

In P4BFT, controllers calculate their decisions in isolation from each other, and transmit them to the destination switch. Control packets are intercepted by the *processing* nodes (i.e., processing switches) responsible for decisions destined for the target switch. In order to collect and compare control packets, we assume packet header fields that include the `client_request_id`, `controller_id`, `destination_switch_id` (e.g., MAC / IP address), the `payload` (controller-decided configuration) and the optional `signature` field (denoting if a packet has already been processed by a processing node). Clients must include the `client_request_id` field in their controller requests.

Apart from distinguishing correct from Byzantine / incorrect messages, P4BFT allows for identification and exclusion of *faulty* controller replicas. In addition to clients, network controllers, and the `REASSIGNER` (as per Section 6.2.4), P4BFT's architectural model also assumes that a subset of available switches is P4-enabled. Depending on the output of `REASSIGNER`'s optimization step, a P4 switch may be assigned the *processing* node role, i.e., become in charge of comparing outputs computed by different controllers, destined for *itself* or *other* configuration targets. A processing node compares messages sent out by different controller replicas and distinguishes the correct ones. On identification of a faulty replica, it declares the faulty replica to the `REASSIGNER`. In P4BFT, `REASSIGNER` is responsible for two tasks:

- *Task 1*: It dynamically reassigns the C2S connections based on the events collected from the detection mechanism of the switches. Namely, it excludes faulty replicas from the assignment

[1] P4 Software Switch - https://github.com/p4lang/behavioral-model

[2] Netronome Agilio®CX 2x10GbE SmartNIC Product Brief - https://www.netronome.com/media/documents/PB_Agilio_CX_2x10GbE.pdf

procedure upon the faulty replica detection. It furthermore ensures that a minimum number of required controllers, necessary to tolerate a F_A Fail-Stop and F_M Byzantine failures, is loaded and associated with each switch. This task is also discussed in Section 6.3.2.

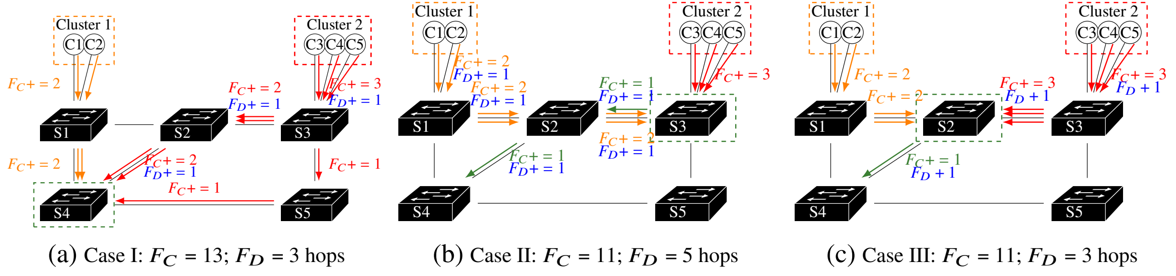
- *Task 2*: It maps a *processing* node, in charge of controller messages' comparison, to each destination switch. Based on the result of this optimization, switches gain the responsibility of control packets processing. The output of the optimization procedure is the *Processing Table*, necessary to identify the switches responsible for comparison of controller messages. Additionally, `REASSIGNER` computes the *Forwarding Tables*, necessary for forwarding of controller messages to processing nodes and reconfiguration targets. Given the number of controllers and the user-configurable parameter of maximum tolerated Byzantine failures F_M, `REASSIGNER` reports to processing nodes the number of *necessary matching messages* that must be collected prior to marking a controller message as correct.

7.2.3 Identification of Optimal Processing Nodes

The extended optimization methodology presented henceforth allows for minimization of the experienced switch reconfiguration delay, as well as the decrease of the total network load introduced by the exchanged controller packets. Note that the presented optimization is orthogonal to MORPH's C2S assignment presented in Section 6.3.2 and Section 7.1.3. When a switch is assigned the *processing* node role for itself or another target switch, it collects the control packets destined for the target switch and deduces the correct payload on-the-fly, it next forwards a single packet copy containing the correct controller message to the destination switch. Consider Fig. 7.7a). If control packet comparison is done only at the target switch (as in prior works), a request for `S4` creates a total footprint of $F_C = 13$ packets in the data plane (the sum of Cluster 1 and Cluster 2 utilizations of 4 and 9, respectively). In contrast, if the processing is executed in `S3` as depicted in Fig. 7.7b), the total experienced footprint can be decreased to $F_C = 11$. Therefore, in order to minimize the total control plane footprint, we identify an optimal processing node for each target switch, based on a given topology, placement of controllers and the processing nodes' capacity constraints. If we additionally extend the optimization to a multi-objective formulation by considering the delay metric, the total traversed critical path between the controller furthest away from the configuration target would equal $F_D = 3$ in the worst case (ref. Fig. 7.7c)), i.e., 3 hops assuming a delay weight of 1 per hop. Additionally, this assignment also has the minimized communication overhead of $F_C = 11$.

Table 7.6: Parameters used in P4BFT's model.

Symbol	Description
$\mathcal{V} : \{S_1, S_2, ..., S_n\}, n \in \mathbb{Z}^+$	Set of all switch nodes in the topology.
$\mathcal{C} : \{C_1, C_2, ..., C_n\}, n \in \mathbb{Z}^+$	Set of all controllers connected to the topology.
$\mathcal{D} : \{d_{i,j,k}, \forall i, j, k \in \mathcal{V}\}$	Set of delay values for path from i to k, passing through j.
$\mathcal{H} : \{h_{i,j}, \forall i, j \in \mathcal{V}\}$	Set of number of hops for shortest path from i to j.
$\mathcal{Q} : \{q_i, \forall i \in \mathcal{V}\}$	Set of switches' processing capacity.
$\mathcal{C}^j \subseteq \mathcal{C}$	Set of controllers connected to the node j.
$\mathcal{M} \subseteq \mathcal{V}$	Set of switches connected to at least one controller.
T	Maximum tolerated delay value.
$x(i, k)$	Binary variable that equals 1 if i is a processing node for k.



(a) Case I: F_C = 13; F_D = 3 hops (b) Case II: F_C = 11; F_D = 5 hops (c) Case III: F_C = 11; F_D = 3 hops

Figure 7.7: For brevity we depict the control flows destined only for configuration target S4. The orange and red blocks represent an exemplary cluster separation of 5 controllers into groups of 2 and 3 controllers, respectively. The green dashed block highlights the processing node responsible for comparing the controller messages destined for S4. Figure (a) presents the unoptimized case as per [1, 115, 128], where S4 collects and processes control messages destined for itself, thus resulting in a control plane load of $F_C = 13$ and a delay on critical path (marked with blue labels) of $F_D = 3$ hops (assuming edge weights of 1). By optimizing for the total communication overhead, the total F_C can be decreased to 11, as portrayed in Figure (b). Contrary to (a), in (b) processing of packets destined for S4 is offloaded to the processing node S3. However, additional delay is incurred by the traversal of path S1-S2-S3-S2-S4 for the control messages sourced in Cluster 1. Multi-objective optimization according to P4BFT, that aims to minimize both the communication overhead and control plane delay instead selects S2 as the optimal processing node (ref. Figure (c)), thus minimizing both F_C and F_D.

We describe the processing node mapping problem using an ILP formulation. Table 7.6 summarizes the notation used.

Communication overhead minimization objective minimizes the global imposed communication footprint in the control plane. Each controller replica generates an individual message sent to the processing node i, that subsequently collects all remaining necessary messages and forwards a resulting single correct message to the configuration target k:

$$M_F = \min \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{V}} (1 \cdot h_{i,k} \cdot x(i,k) + \sum_{j \in \mathcal{M}} |C^j| \cdot h_{j,i} \cdot x(i,k)) \tag{7.2}$$

Configuration delay minimization objective minimizes the *worst-case* delay imposed on the critical path used for forwarding configuration messages from a replica associated with node j, to the potential processing node i and finally to the configuration target node k:

$$M_D = \min \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{V}} x(i,k) \cdot \max_{\forall j \in \mathcal{M}} (d_{j,i,k}) \tag{7.3}$$

Bi-objective optimization minimizes the weighted sum of the two objectives, w_1 and w_2 being the associated weights:

$$\min w_1 \cdot M_F + w_2 \cdot M_D \tag{7.4}$$

Processing capacity constraint: Sum of messages requiring processing on i, for each configuration target k assigned to i, must be kept at or below i's processing capacity q_i:

$$\text{Subject to:} \quad \sum_{k \in \mathcal{V}} x(i,k) \cdot |C| \leqslant q_i, \quad \forall i \in \mathcal{V} \tag{7.5}$$

Maximum delay constraint: For each configuration target k, the delay imposed by the controller packet forwarding to node i, responsible for collection and packet comparison procedure and forwarding

of the correct message to the target node k, does not exceed an upper bound T:

$$\text{Subject to:} \quad \sum_{i \in \mathcal{V}} x(i,k) \cdot \max_{\forall j \in \mathcal{M}} (d_{j,i,k}) \leqslant T, \quad \forall k \in \mathcal{V} \tag{7.6}$$

Single assignment constraint: For each configuration target k, there exists *exactly* one processing node i:

$$\text{Subject to:} \quad \sum_{i \in \mathcal{V}} x(i,k) = 1, \quad \forall k \in \mathcal{V} \tag{7.7}$$

Note: The assignment of C2S connections for the purpose of control and reconfiguration is reused from Section 6.3.2 and Section 7.1.3 and is thus not detailed here.

7.2.4 P4 Switch and `REASSIGNER` Control Flow

Processing node (Data Plane): Switches declared to process controller messages for a particular target (i.e., for itself, or for another switch) initially collect the control payloads stemming from different controllers. Each processing node maintains counters for the number of observed and matching packets for a particular (re-)configuration request identifier. After sufficient matching packets are collected for a particular payload (more specifically, hash of the payload), the processing node *signs* a message using its private key and forwards one copy of the correct packet to its own control plane for required software processing (i.e., identification of the correct message and potentially Byzantine controllers), and the second copy on the port leading to the configuration target. To distinguish processed from unprocessed packets in destination switches, processing nodes refer to the trailing signature field.

Processing node (Control Plane): After determining the correct packet, the processing node identifies any incorrect controller replicas (i.e., replicas whose output hashes diverge from the deduced correct hash) and subsequently notifies the `REASSIGNER` of the discrepancy. Alternatively, the switch applies the configuration message if it is the configuration target itself. The switch then proceeds to clear its registers associated with the processed message hash so to free the memory for future requests.

`REASSIGNER` *control flow:* At network bootstrapping time, or on occurrence of any of the following events: i) a detected Byzantine controller; ii) a failed controller replica; or iii) a switch/link failure; `REASSIGNER` reconfigures the processing and forwarding tables of the switches, as well as the number of required matching messages to detect the correct message.

7.2.5 P4 Tables Design

Switches maintain *Tables* and *Registers* that define the method of processing incoming packets. `REASSIGNER` populates the switches' Tables and Registers so that the selection of processing nodes for controller messages is optimal w.r.t. a set of given constraints, i.e., so that the total message overhead or control plane latency experienced in control plane is minimized (according to the optimization procedure in Section 7.2.3). The `REASSIGNER` thus modifies the elements whenever a controller is identified as incorrect and is hence excluded from consideration, resulting in a different optimization result. P4BFT leverages four P4 tables:

Table 7.7: Hash table layout of a P4BFT's processing node.

Msg Hash	**Request ID** 1				...	**Request ID** K			
h_0	$b_{C_1}^{h_0}$	$b_{C_2}^{h_0}$	...	$b_{C_N}^{h_0}$	...	$b_{C_1}^{h_0}$	$b_{C_2}^{h_0}$	...	$b_{C_N}^{h_0}$
...	$b_{C_1}^{\cdots}$	$b_{C_2}^{\cdots}$	...	$b_{C_N}^{\cdots}$	...	$b_{C_1}^{\cdots}$	$b_{C_2}^{\cdots}$	...	$b_{C_N}^{\cdots}$
h_{F_M}	$b_{C_1}^{h_{F_M}}$	$b_{C_2}^{h_{F_M}}$	...	$b_{C_N}^{h_{F_M}}$	...	$b_{C_1}^{h_{F_M}}$	$b_{C_2}^{h_{F_M}}$	...	$b_{C_N}^{h_{F_M}}$

1. *Processing Table*: It holds identifiers of the switches whose packets must be processed by the switch hosting this table. Incoming packets are matched based on the destination switch's identifier. In the case of a table hit, the hosting switch processes the packets as a processing node. Alternatively, the packet is matched against the *Process-Forwarding Table*.

2. *Process-Forwarding Table*: Declares which egress port the packets should be sent out on for further processing. If an unprocessed control packet is not to be processed locally, the switch will forward the packet towards the correct processing node, based on forwarding entries maintained in this table.

3. *L2-Forwarding Table*: After the processing node has processed the incoming control packets destined for the destination switch, the last step is forwarding the correctly deduced packet towards it. Information on how to reach the destination switches is maintained in this table. Contrary to forwarding to a processing node, the difference here is that the packet is now forwarded to the destination switch.

4. *Hash Table with Associated Registers*: *Processing* a set of controller packets for a particular request identifier requires evaluating and counting the number of occurrences of packets containing the matching payload. To uniquely identify the decision of the controller, a hash value is generated on the payload during processing. The counting of incoming packets is done by updating the corresponding binary values in the register vectors, with respective layout depicted in Table 7.7.

On each arriving unprocessed packet, the processing node computes a previously seen or i-th initially observed hash $h_i^{request_id}$ over the acquired payload. Subsequently, it sets the binary flag to 1, for source controller `controller_id` in the i-th register row at column [`client_request_id` $\cdot\, |C| +$ `controller_id`]. $|C|$ represents the total number of deployed replicas. Each time a client request is fully processed, the binary entries associated with the corresponding `client_request_id` are reset to zero. To detect a Byzantine controller, the replica identifiers associated with hashes distinguished as incorrect, are reported to the `REASSIGNER`.

Note: To tolerate F_M Byzantine failures, a maximum of $F_M + 1$ unique hashes for a single request identifier may be detected, hence the corresponding $F_M + 1$ pre-allocated table rows in Table 7.7.

7.2.6 Evaluation & Results

We next evaluate the following metrics using P4BFT and state-of-the-art [1, 115, 128] designs: i) control plane load; ii) imposed processing delay in the software and hardware P4BFT nodes; iii) E2E switch reconfiguration delay; and iv) ILP solution time. We execute the measurements for random

controller placements and diverse data plane topologies: i) random topologies with fixed average node degree; ii) reference Internet2 [217]; and iii) data-center Fat-Tree ($k = 4$). We also vary and depict the impact of number of switches, controller instances, and disjoint controller clusters. To compute C2S paths and paths between processing and destination switches, `REASSIGNER` leverages the CSPF algorithm. For brevity, as an input to the optimization procedure in `REASSIGNER`, we assume edge weights of 1. The objective function used in processing node selection is Eq. 7.4, parametrized with $(w_1, w_2) = (1, 1)$.

P4BFT implementation is a combination of $P4_{16}$ and P4 Runtime code, compiled for software and physical execution on P4 software switch *bmv2* (*master* check-out, December 2018) and a *Netronome Agilio SmartNIC* device with the corresponding firmware compiled using `Software Development Kit (SDK) 6.1-Preview`, respectively. Apache Thrift and gRPC Remote Procedure Calls (gRPC) are used for population of registers and table entries in *bmv2*, respectively. Thrift is used for both table and registers population for the *Netronome SmartNIC*, due to the current SDK release not fully supporting the P4 Runtime. Hypertext Transfer Protocol (HTTP) REST is used in exchange between P4 switch control plane and the `REASSIGNER`. The `REASSIGNER` and controllers are implemented in Python.

7.2.6.1 Communication Overhead Advantage

Fig. 7.8 depicts the packet load improvement in P4BFT over MORPH and other reference solutions [115, 128] for randomly generated topologies with average node degree of 4. The footprint improvement is defined as $1 - \frac{F_C^{P4BFT}}{F_C^{SoA}}$, where F_C denotes the sum of packet footprint for control flows destined to each destination switch of the network topology as per Section 7.2.3 and Fig. 7.7. P4BFT outperforms the state of the art as each of the presented works assumes an uninterrupted control flow from each controller to the destination switches. P4BFT, on the other hand, aggregates control packets in the processing nodes that, subsequently to collecting the control packets, forward a single correct message towards the destination, thus decreasing the control plane load.

Fig. 7.9 (a) and (b) portray the footprint improvement scaling with the number of controllers and disjoint clusters. P4BFT's footprint efficiency generally benefits from the higher number of replicas. Controller clusters, on the other hand, aggregate replicas behind the same edge switch. Thus, with the higher number of disjoint clusters, the degree of aggregation and the total footprint improvement decreases.

7.2.6.2 Processing and Reconfiguration Delay

Fig. 7.10 depicts the processing delay incurred in the processing node for a single client request. The delay corresponds to the P4 pipeline execution time spent on identification of a correct controller message, comprising the i) hash computation over controller messages; ii) incrementing the counters for the computed hash; iii) signing the correct packet and; iv) propagating it to the correct egress port. When using the P4-enabled SmartNIC, P4BFT decreases the processing time compared to *bmv2* software target by two orders of magnitude.

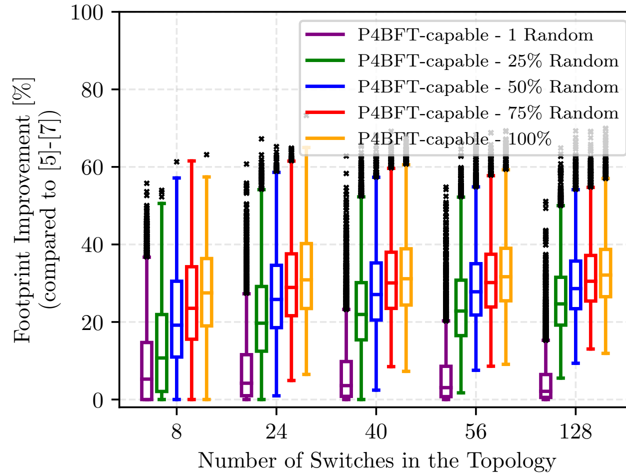


Figure 7.8: Packet load improvement of P4BFT over the reference works [1, 115, 128] for 5000 randomly generated network topologies per scenario, with 7 controllers distributed into 3 disjoint and randomly placed clusters. In addition to the 100% coverage where each node may be considered a P4BFT processing node, we include scenarios where only the random $[1, 25\%, 50\%, 75\%]$ nodes of all available nodes in the infrastructure are P4BFT-enabled. Thus, even in the topologies with limited programmable data plane resources, i.e., in brownfield-scenarios involving OF / NETCONF + YANG non-P4 configuration targets, P4BFT offers substantial advantages over existing state of the art.

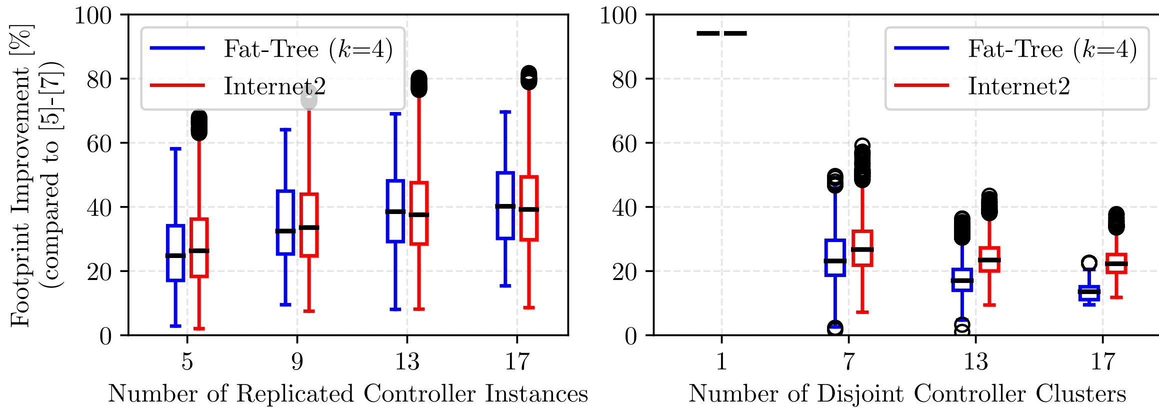


Figure 7.9: The impact of (a) controllers and; (b) disjoint controller clusters on the control plane load footprint in Internet2 and Fat-Tree ($k = 4$) topologies for 5000 randomized controller placements each. (a) randomizes the placement but fixes the number of disjoint clusters to 3; (b) randomizes the number of disjoint clusters between $[1, 7, 13, 17]$ but fixes the number of controllers to 17. The resulting footprint improvement scales with the number of controllers but is inversely proportional to the number of disjoint clusters.

Fig. 7.11 depicts the total reconfiguration delay imposed in state of the art and P4BFT designs for $(w_1, w_2) = (1, 1)$ (ref. Eq. 7.4). It considers the time difference between issuing a switch reconfiguration request, until the correct controller message is determined and applied in the destination. Related works process the reconfiguration messages stemming from controller replicas in the destination target, their control flows traversing shortest paths in all cases. On average, P4BFT's reconfiguration delay is comparable with related works, the overall control plane footprint being substantially improved.

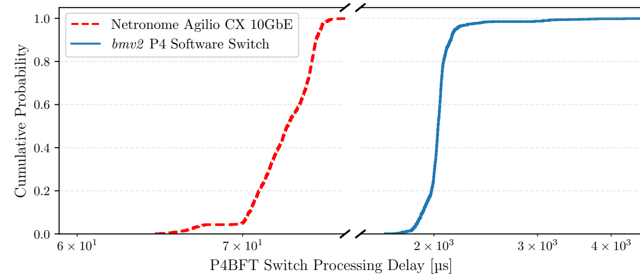


Figure 7.10: The ECDFs of processing delays imposed in a P4BFT's processing node for a scenario including 5 controller replicas. 3 correct packets and thus 3 P4 pipeline executions are necessary to confirm the payload correctness when tolerating 2 Byzantine controllers.

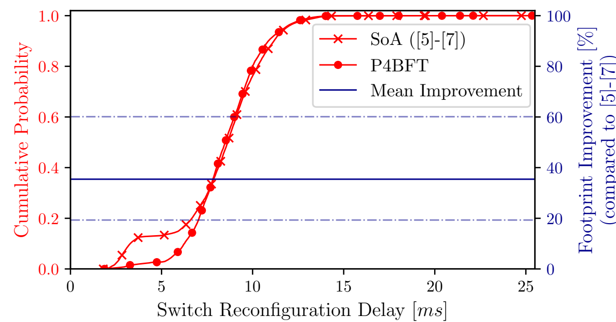


Figure 7.11: ECDFs of time taken to configure randomly selected switches in state-of-the-art and P4BFT environments for Internet2 topology, 10 random controller placements for 5 replicas and 1700 individual requests per placement. State-of-the-art works collect, compare and apply the controllers' reconfiguration messages in the destination switch thus effectively minimizing the reconfiguration delay at all times. P4BFT, on the other hand, may occasionally favor footprint minimization over the incurred reconfiguration delay and thus impose a longer critical path, leading to slower reconfigurations. On average, however, P4BFT imposes comparable reconfiguration delays at a much higher footprint improvement (depicted blue), mean being 38%, best and worst cases at 60% and 19.3%, respectively, for evaluated placements.

7.2.6.3 Impact of Optimization Objectives

Fig. 7.12 depicts the Pareto frontier of optimal processing node assignments w.r.t. the objectives presented in Section 7.2.3: the total control plane footprint (minimized as per Eq. 7.2) and the reconfiguration delay (minimized as per Eq. 7.3). From the total solution space, depending on the weights prioritization in Eq. 7.4, either $(26.0, 3.0)$ or $(28.0, 2.0)$ solutions can be considered optimal. Comparable works implicitly minimize the incurred reconfiguration delay but fail to consider the control plane load. Hence, they prefer the $(30.0, 2.0)$ solution (encircled red).

7.2.6.4 ILP Solution Time - Impact of Topology And Controller Clusters

The solution time for the optimization procedure considering random topologies with average network degree of 4 and a fixed number of randomly placed controllers is depicted in Fig. 7.13 (a). The solution time scales with number of switches, peaking at $420ms$ for large 128-switch topologies. The reassignment procedure is executed in few rare events: during network bootstrapping, on Byzantine / unavailable controller detection and following a switch / link failure. Thus, we consider the observed

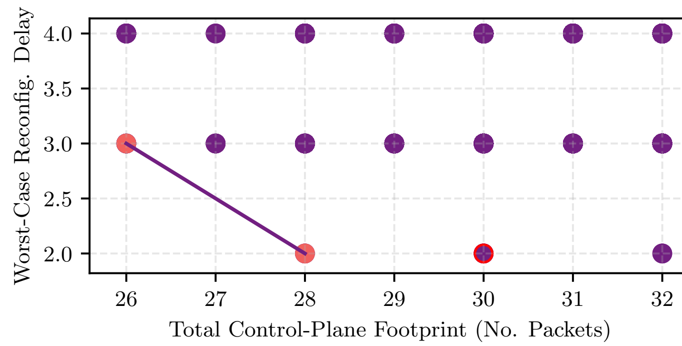


Figure 7.12: Pareto Frontier of P4BFT's solution space for the topology presented in Fig. 7.7. The comparable works tend to minimize the incurred reconfiguration delay, but ignore the imposed control plane load. [1, 115, 128] hence select (30.0, 2.0) as the optimal solution (encircled in red) while P4BFT selects (26.0, 3.0) or (28.0, 2.0) thus minimizing the total overhead as per Eq. 7.4.

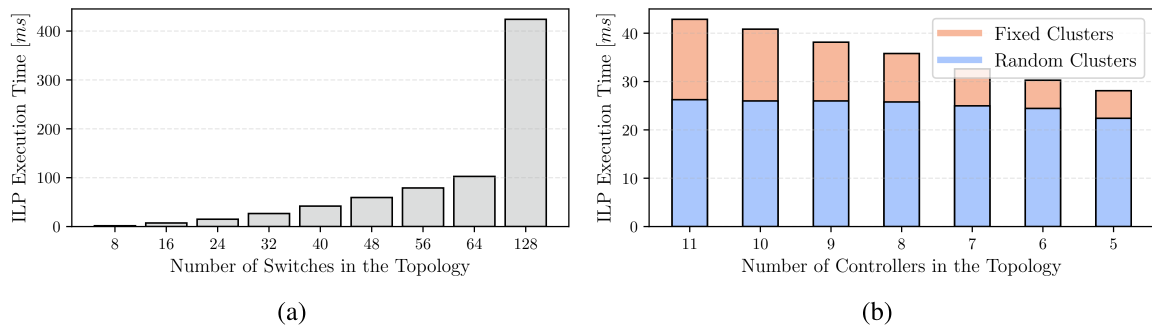


Figure 7.13: (a) depicts the impact of network topology size on the ILP solution time for random topologies. (b) depicts the impact of controller number and cluster disjointness in the case of Internet2 topology. The results are averaged over 5000 per-scenario iterations. The higher the cluster aggregation of controllers, the lower the ILP solution time. "Fixed Clusters" considers the worst-case, where each controller is randomly placed, but disjointly w.r.t. the other controller instances. Clearly, the ILP solution time scales with the amount of deployed switches and controllers. We used Gurobi 8.1 optimization framework configured to execute multiple solvers on multiple threads simultaneously, and have chosen the ones that finish first.

solution time short and viable for online mapping. Fig. 7.13 (b) depicts the ILP solution time scaling with the number of active controllers. The lower the number of active controllers, the shorter the solution time. In "Fixed Clusters" case, each controller is placed in its disjoint cluster (worst-case for the optimization). The "Random Clusters" case considers a typical clustering scenario, where a maximum of [1..3] clusters are deployed, each comprising a uniform number of controller instances. The higher the cluster aggregation, the lower the ILP solution time.

7.3 Chapter Summary and Outlook

In the first part of the chapter, we have proposed and implemented two agreement-based and an opportunistic BFT protocol for the purpose of client request sequencing and state synchronization, and have analyzed their overheads in an emulated environment using emulated well-known production topologies. The evaluated KPIs include the switch reconfiguration times, the request acceptance rates and the communication overhead. We have shown how our opportunistic BFT approach leverages agreement

of switch state at the time of request computation to ensure the causality during request reconfiguration. It offers considerably lower response time compared to the sequencing-based approaches. However, this benefit comes at the expense of a lower acceptance rate and quadratic communication overhead. For those metrics, the A&E group-based sequencing approach SBFT presents a better alternative. Both approaches, however, result in a higher throughput compared to the state-of-the-art MPBFT, which adapts the well-established PBFT protocol.

In the second part of the chapter, we introduced P4BFT - a switch control- / data-plane co-design, capable of Byzantine controller identification while simultaneously minimizing the control plane footprint. By merging the control channels in P4-enabled *processing* nodes, the use of P4BFT results in a lowered control plane footprint, compared to basic MORPH design as well as the state-of-the-art works. In a hardware-based data plane, by offloading packet processing from general purpose CPU to the data-plane Network Processing Unit (NPU), it additionally leads to a decrease in request processing time. Given the low solution time, the presented ILP formulation for processing node identification is viable for online execution.

Outlook: While we focused on an SDN scenario here, future works should consider the conceptual transfer of P4BFT's in-network offloading and acceleration to other application domains, including stateful web applications and critical industrial control systems that rely on BFT operation [62, 103, 179, 227].

Chapter 8

On Deployment Complexity of Distributed Control Plane

Initial SDN deployments (e.g., in data-centers and Wide Area Networks (WANs) [73]) have relied on Out-Of-Band Network Control (OOBC) with which control traffic is exchanged between switches and the SDN controller(s) using dedicated links and switch management ports. Bootstrapping with OOBC is trivial, due to controllers being capable of controlling the switches without requiring connectivity over any other switches. Successful adoption of SDN in critical industrial environments will, however, require In-Band Network Control (IBC). The OOBC model is inapplicable due to its associated Capital Expenses (CAPEX) and installation efforts [38, 82, 176]. For example, deploying an additional physical management network in machine tool manufacturing or drive technology networks is immensely expensive, due to associated cabling costs for the shielding required for RF / EMI noise suppression. Similarly, avionics and automotive networks benefit greatly from decreased cabling weight [82]. Existing designs for seamlessly enabling IBC, however, cater only for single-controller operation, assume proprietary switch modifications, and / or require a high number of manual configuration steps, making them non-resilient to failures and hard to deploy.

To address these concerns and ease the deployments of the presented distributed SDN control plane designs, we introduce two *nearly* completely automated bootstrapping schemes for a multi-controller IBC resilient to link, switch, and controller failures. One assumes hybrid OF / legacy switches with deployed (Rapid) Spanning Tree Protocol ((R)STP) and the second uses an incremental approach that circumvents (R)STP. We implement both schemes as ODL extensions, and qualitatively evaluate their performance with respect to: the time required to converge the bootstrapping procedure; the time required to dynamically extend the network; and the resulting flow table occupancy. The proposed schemes enable fast bootstrapping of a robust, IBC industrial networks with support for seamless redundancy of control flows and network extensions, while ensuring interoperability with off-the-shelf switches.

The first of the two presented schemes was demonstrated successfully in an operational industrial network with critical fail-safe requirements [9]. Both schemes were evaluated in emulated small- and large-scale industrial topologies. *The designs of the two schemes were published in augmented form in our publication* [5].

8.1 Motivation for In-Band Bootstrapping of a Reliable Distributed Control Plane

With IBC, control traffic is forwarded along the same links used for the data plane traffic. This makes IBC networks the preferable solution from the cost and operational perspectives. However, their implementation is challenging. For example, current ODL [123] and ONOS [39] releases provide no mechanisms to support or automate the IBC bootstrapping, nor do they allow for protection of control plane traffic against arbitrary link and node failures, both which are of critical importance in industrial networks [204].

Summarized, the bootstrapping of a *reliable* softwarized industrial network must address the following challenges:

(1) **Establishing robust S2C and C2C communication using IBC and an initially unconfigured control plane**: In industrial networks, OOBC imposes a requirement for ruggedized wiring immune to EMI and heavy electrical surges typical for electric utility substations, factory floors and traffic control cabinets, further raising the costs. Hence, IBC is the preferable way of controlling SDN switches. In contrast to OOBC, IBC requires a more complex initial setup, due to: i) unpredictable data plane traffic that could impact the control plane responsiveness; ii) data plane failures which may impact the availability of the control plane; iii) establishing deterministic control plane flows in a non-configured data plane is non-trivial, and typically requires manual configuration.

(2) **Preserving the control plane stability in case of arbitrary number of link, switch and/or controller failures**: To protect against controller's SPOF, state-of-the-art controllers deploy and replicate their state across multiple replicas in SC, EC or AC manner (ref. Chapter 5) [3, 113, 150]. ODL [123] and ONOS [39] leverage the SC model relying on Raft [87].

Ensuring consistent topology and switch state views relies on C2C communication, thus making it crucial for bootstrapping procedure to enable robust control channels by design. The physical dislocation of controllers additionally imposes transmission of the synchronization traffic using IBC flows. ODL's southbound module (*OpenFlowPlugin*) requires a prior C2C channel establishment and a successful consensus round even for a "trivial" population of OF rules during bootstrapping. Similarly, the per-switch controller's role (i.e., "backup" or "master") must be decided using consensus among the contending controllers. Hence the configuration of switches for multi-controller support and the support for controller state synchronization must be targeted in tandem.

Enabling a resilient distributed control plane thus requires the bootstrapping processes to ensure resilience in case of following failures:

- Failures of F out of $2F+1$ controllers in the case of a Fail-Stop control plane (as per Chapter 3): No managed switches may be left unmanaged due to individual replica failures;
- Failures of up to F_M and F_A out of $2F_M + F_A + 1$ controllers in the case of a BFT control plane (as per Chapter 6);

- Assuming $k + 1$ disjoint paths between each two controllers, failure of k switches / links must not result in control plane partitioning [68, 201].

(3) **Support for node and link extensions of the previously reliably bootstrapped IBC network**: Dynamic network topology changes, i.e., attachment of new end-devices (host discovery) and packet forwarding devices to existing managed networks should be natively supported by the proposed designs. In particular, dynamic plugging and unplugging of machine networks (i.e., network cells) in industrial backbones, reconfiguration of modular machine network assemblies and addition / removal of production line networks are typical Industry 4.0 use cases [213, 224].

(4) **Minimization of manual per-device preconfiguration**: Involved configuration effort should be minimized - Configuration of controller lists (comprising IP address / port number pairs) and switch identities should be automated so to enable agile device deployment / network extensions (e.g., for VNF on-boarding). To this end, controllers or external Dynamic Host Configuration Protocol (DHCP) servers should be charged with providing switches with management IP addresses and thus automated identity binding.

In the remainder of the chapter, we tackle the challenge of bootstrapping a reliable IBC softwarized network while satisfying challenges (1)–(4):

- We propose two novel automated bootstrapping schemes targeting reliable IBC networks with multi-controller support and solving the above-mentioned challenges.
- We evaluate the proposed designs in realistic industrial topologies with varying sizes and controller placements w.r.t. *(i)* the time required to converge the bootstrapping procedure, *(ii)* the time required to dynamically extend the network, and *(iii)* the flow table occupancy. We limit our evaluation to the two schemes proposed in this paper, due to the unavailability and / or ambiguity of closed-source state-of-the-art solutions.
- We discuss implementation aspects relevant for the successful introduction of the designs in networks equipped with off-the-shelf OF agents.

8.2 Related Work - On In-Band Bootstrapping of Distributed Control Plane

The few automated IBC bootstrapping approaches to date address the above mentioned challenges partially or in isolation of each other. For instance, they neglect the SPOF issue [175–177], do not guarantee reliable control connections [167, 180], omit the practical constraint of controllers' state synchronization prior to switch reconfigurations [51, 167, 168], or assume functional extensions of switch components, e.g., the DHCP client or OF agent [38, 176, 177]. This limits their applicability in existing infrastructures. Most concerning, the reproducibility of state-of-the-art proposals in real environments is limited, due to unclear preconfiguration assumptions or closed-source implementations.

Design	Auto. Switch Management IP Provisioning	Auto. Controller List Provisioning	Resilience of Control Flows	Multi-Controller Support	(R)STP Not Required	No Proprietary Switch Extensions
Sharma et al. [175–177]	✓	✓	✓ / ✗ (reactive)	✗	✓	✗
Schiff et al. I [167–169]	✗	✗	✓ / ✗ (timeout)	✓	✓	✓
Schiff et al. II [50]	?	?	✓ (timeout/reactive)	✓	✓	✓
Heise et al. [82]	✗	✗	✓ (proactive replication)	✗	✗	✓
Canini et al. [51]	✗	✗	✓ (reactive)	✓	?	✓
Su et al. [180]	✗	✓	✓ (reactive)	✗	?	✓
Bentstuen et al. [38]	✓	✓	✗	✗	✓	✗
Hybrid Switch Scheme (HSS)	✓	✓	✓ (proactive replication)	✓	✗	✓
Hop-by-Hop Scheme (HHS)	✓	✓	✓ (proactive replication)	✓	✓	✓

Table 8.1: Comparison of our HSS and HHS bootstrapping designs with existing schemes.

Sharma et al. [175–177] were first to propose an automatic bootstrapping scheme and evaluate its performance for various IBC topologies. [175, 177] highlight the advantages of a proactive protection scheme (using fast-failover groups [29]), allowing the controller to proactively compute duplicate paths for control plane flows. On a successful failure discovery, the detecting switch automatically re-routes the incoming traffic over the assigned backup port, without needing to involve the controller in loop. In all three works, Sharma et al. assume proprietary modifications to the DHCP client hosted in switches with the goal of provisioning the controller lists. Distributed control plane was not considered by these works.

Schiff et al. [168] present a design of a self-organizing multi-controller control plane that relies exclusively on OF. Contrary to the Hybrid Switch Scheme (HSS) and Hop-by-Hop Scheme (HHS) schemes presented here, the authors do not consider the necessity of controller state synchronization prior to switch reconfigurations. In [167], the authors extend their approach to include a timeout-based fault-tolerance approach where rules corresponding to failed paths *eventually* time out, thus preventing permanent switch cut-offs. Instead, we propose constant duplication of control flows incurring zero-packet-loss in case of failures. Follow-up works [50, 51] propose a timeout-free approach to ensure resilience against data plane failures, based on assumption of a controller-initiated switch discovery and OF *equal role* [122] controller association with switches. Similar to above works, we compute and iteratively expand the spanning tree so to enable loop-less forwarding of control traffic and discovery of newly added switches to an already bootstrapped network.

[169] proposes atomic transactions for coordinated concurrent switch configurations by multiple controllers. The approach is orthogonal to our work but we additionally assume the requirement for distributed consensus [3, 87], as imposed by ODL [123] and ONOS [39] implementations.

Heise et al. [82] propose the usage of network calculus, i.e., rate- and burst-policed control traffic for providing upper bound guarantees for bootstrapping convergence time. They leverage fast-failover groups to implement the restoration of control flows in face of failures. Their bootstrapping concept assumes (R)STP in switches and no multi-controller support. Bentstuen et al. [38] propose an approach that relies on intent-based control flow definitions targeting ONOS [39], so to simplify the management of control flows in a *single-controller* environment. The authors also stumble upon a number of practical issues related to IBC bootstrapping and ultimately fall back to modification of the OVS's source code. To support the existing OF implementations, we considered workarounds for these limitations.

FASIC [180] minimizes congestions in single-controller IBC control plane by means of a centralized control port load monitoring. Control port is switched to a more fitting port on exceeded threshold

or in case of link failures. The authors deduce that their fail-over approach results in non-negligible packet loss, related to OF's back-off interval in the case of repeated unsuccessful connection attempts. We consider this aspect in the design of the (R)STP timer in the HSS approach. In [72], authors propose a method for re-routing IBC control flows based on observed controller load and IBC channel congestion. To this end, the authors leverage control flow *shifting* and *splitting*. Both methods are orthogonal and compatible with HSS and HHS.

8.3 Background and Terminology

8.3.1 General Model

A *failed* controller replica is an inactive, unreachable replica. Apart from individualized identity configuration, each replica is an exact clone and thus implements the OF southbound logic and is capable of executing the bootstrapping procedure. Updates to a replica's distributed data store are synchronized among all reachable and active replicas using Raft. Thus, the described bootstrapping procedure is tolerant against Fail-Stop failures only.

Note: Tolerating Byzantine failures is currently unsupported and is considered as future work.

Switches are operated in OF Master-Slave mode [228], where any controller replica may become the Master of a switch. Forwarding table modification messages initiated by a Slave controller are proxied through the switch's Master. All switches are controlled in IBC manner, using OF v1.3+ [228]. To enable the (optional) automated identity (IP address) and controller list assignment, each switch deploys a DHCP client and a Secure Shell (SSH) server. The controllers accordingly execute DHCP server instances for automated IP address roll-out. For purpose of automated authentication, prior to bootstrapping step, switches and controllers may have their certificates or symmetric keys on-boarded so to enable Public Key Infrastructure (PKI)-, or Message Authentication Code (MAC)-based authentication [59, 65], respectively. Depending on the target scheme, the switches either all support (R)STP or they do not (i.e., they have it disabled). Moreover, each switch is capable of forwarding traffic via traditional MAC learning (using `NORMAL`-mode forwarding). We assume an Internet Protocol v4 (IPv4)-enabled infrastructure.

A k link / switch fault resilience model requires an adequate underlying topology, where $k + 1$ disjoint paths can be found for any C2S / C2C pair. For simplification, we relax this condition and model a link failure between an SDN controller and the switch directly attached to it as a controller failure. We duplicate traffic on disjoint paths for each communication pair. Higher layer protocols are then tasked with duplicate elimination, i.e., we rely on TCP at transport layer. This is feasible, since both OF and C2C synchronization in ODL transmit TCP traffic. A generalized solution should instead deploy robust duplicate frame elimination measures (e.g., as per TSN 802.1CB [154]).

8.3.2 Secure and Standalone Modes

In *Standalone* mode, a switch forwards packets autonomously - i.e., it behaves like a non-managed MAC learning forwarder. Namely, it contains one *generic* OF rule that matches all traffic and forwards

it to the reserved `NORMAL` (ref. [122]) port. After establishing a connection to a controller, the `NORMAL` rule is automatically removed.

In *Secure* mode, it is non-existent. Following an established controller connection in Secure mode, the flow table remains unmodified, and the forwarding behavior continues according to controller-added rules. In case of an empty flow table and non-configured controller lists, the switch in Secure mode drops all arriving traffic. Depending on the scheme, switches must support either *Secure and Standalone* mode or *Secure* mode only.

8.3.3 In-band Mode

With *In-band* mode, a switch is initialized with *generic* rules that ensure controller-destined traffic is forwarded according to the MAC learning table (i.e., using the `NORMAL` port).

In switches basing their OF implementation on the OVS agent, the In-band rules have a priority higher than any flow rules that may be configured by the controller. There, In-band mode automatically preconfigures the rules for forwarding: i) Address Resolution Protocol (ARP) requests and DHCP Discover messages generated by the switch; and ii) ARP replies and DHCP Offer messages destined for the switch. For each controller, rules matching following traffic are added: i) ARP replies destined for controller's MAC address; ii) ARP requests generated with controller's MAC address as source; iii) ARP replies containing controller's IP address as target; iv) ARP requests containing controller's IP address as source; v) traffic destined for controller's IP / TCP port pair; and vi) traffic with controller's IP and TCP port as source.

8.4 HSS - (R)STP-based Automated Distributed Control Plane Bootstrapping

We first introduce the HSS bootstrapping scheme that heavily relies on existence of (R)STP, Standalone, In-band modes, and `NORMAL` forwarding. HSS leverages (R)STP to establish an acyclic graph and thus remedy the potential storm issues stemming from traffic broadcasts (e.g., from ARP and DHCP requests). Fig. 8.1 summarizes the workflow of HSS. In `Phase 1a` and `Phase 1b`, controllers establish the C2C communication over the spanning tree computed by the network. In `Phase 1c`, switches are provided dedicated management IP addresses and controllers' IP / port pairs. In `Phase 2a`, controllers establish the control over the connected switches, install a set of initial rules and eventually disable (R)STP in switches so to enable blocked ports. Explicit resilient flow rules are embedded in `Phase 2b`, so to fulfill the fault-tolerance requirements. To support network extensions at runtime (`Phase 3`, not depicted), a virtual spanning tree is maintained in controllers and enforced for discovery traffic, allowing for safe forwarding of (broadcast) traffic initiated by newly attached nodes, even after disabling (R)STP.

HSS requires the following assumptions to hold at startup: i) Controllers are aware of IP addresses of other controllers, or are capable of resolving these using standardized Domain Name System (DNS)

queries; ii) Switches are initialized in Standalone mode with (R)STP and In-band mode enabled; iii) Switches are provisioned with controllers' public certificates or symmetric MAC keys [59, 65].

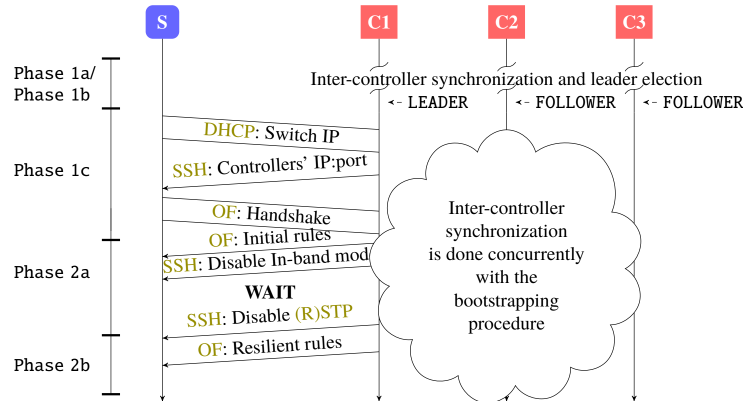


Figure 8.1: HSS - Message sequence diagram of the bootstrapping procedure as described in Section 8.4.

8.4.1 Phase 1: Network Startup and Initial Address Distribution

8.4.1.1 Phase 1a - In-Band Flow Rules

Switches boot with In-band flow rules set up, enabled to bidirectionally forward traffic between switches and controllers. The traffic matching In-band flow rules is forwarded using the reserved `NORMAL` OF port. After a switch establishes a connection with the controller, due to enabled Standalone mode, the switch automatically deletes any rules which are of a priority lower than the In-band rules. Without enabling the In-band mode, this behavior would lead to switch flow tables becoming empty, and thus traffic drops, including the controller-initiated OF traffic. Instead, In-band mode rules (ref. Section 8.3.3) ensure the incoming traffic is forwarded even after a successful initial C2S connection.

8.4.1.2 Phase 1b - Controller Synchronization

The switches initially boot with Standalone mode and (R)STP enabled. Prior to taking the switch ownership, the controllers elect a leader and establish an active Raft cluster using the established spanning tree thus enabling synchronization.

8.4.1.3 Phase 1c - Distribution of Switch and Controller Connection Identifiers

As mentioned before, a fully-automated address distribution to the switches' management interfaces is an optional feature of our schemes. To this end, a DHCP server instance, e.g., realized as a replicated application of the controller, distributes the IPv4 addresses to switches on an First-Come-First-Serve (FCFS) basis. To omit potential address conflicts, the Raft leader replica is charged with address leasing using an automatically derived list of free addresses given the reserved controller IP addresses. Follower replicas on the other hand, ignore DHCP requests, as long as the current leader is active. In case of duplicate DHCP requests, DHCP server replies only to the first request [176].

The leader then establishes management session with provisioned switches (using SSH / Open vSwitch Database Protocol (OVSDB)). Since in this phase all switches forward control traffic according to the `NORMAL` data-path (Standalone mode) the connection establishment succeeds. The leader next proceeds to provision the switches with redundant controllers' IP addresses and TCP ports.

8.4.2 `Phase 2`: Enabling a Functional and Resilient Control Plane

The goal of `Phase 2` is to provide the discovered switches with rules necessary to enable a resilient control plane. `Phase 2` is divided into two sub-steps: `Phase 2a` - initial control plane rules are installed to allow communication with adjacent switches; and `Phase 2b` - resilient control plane rules are installed. To avoid broadcast storms, (R)STP remains functional until all initially booted switches are discovered. (R)STP is disabled in `Phase 2b` and, after all links were discovered, Raft leader computes and embeds resilient paths.

8.4.2.1 `Phase 2a` - Initial OpenFlow Flow Rules

For each connected switch the leader controller installs the rules depicted in Table 8.2. The matching semantics correspond to OF fields defined in [228]. The leader next disables the In-band mode flow rules using the SSH / OVSDB management channel. This is done in order to enable full control over control plane traffic forwarding based on initial rules. Otherwise, e.g., in OVS, the switches would continue to forward OF, ARP and DHCP traffic according to In-band mode imposed rules, since they have the highest available priority.

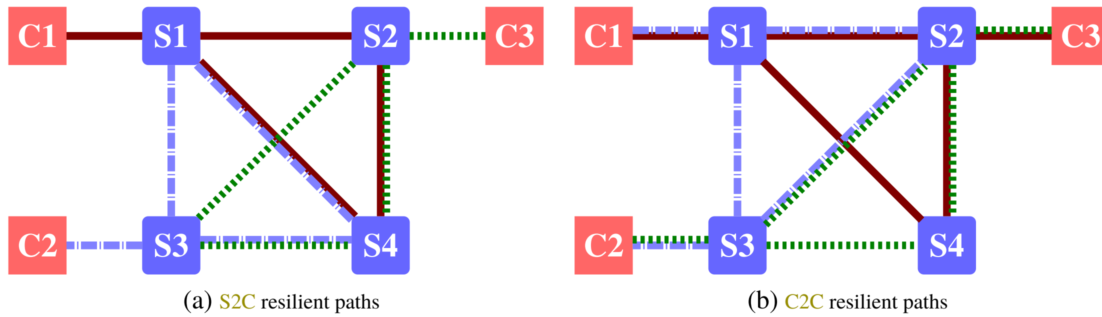
On rule semantics (Table 8.2): DHCP and SSH rules enable a continued successful execution of `Phase 1` for switches adjacent to the configured switch. ARP rules prevent the ARP table cache timeouts. Link Layer Discovery Protocol (LLDP) rules allow the controllers to discover topology updates. An additional per-controller ARP rule is required for controller's placement discovery. To this end, controllers periodically generate probe packets.

Purpose	Packet Type	Matching	Action
Dynamic switch IP address configuration	DHCP	udp, udp_src=68 udp, udp_src=67	Send to `NORMAL`
Remote switch configuration	SSH	tcp, tcp_src=22 tcp, tcp_src=22	Send to `NORMAL`
C2S OF interaction	OF	tcp, tcp_src=6653 tcp, tcp_dst=6653	Send to `NORMAL`
Switch IP resolution	ARP	arp, arp_tpa=control plane IP prefix arp, arp_spa=control plane IP prefix	Send to `NORMAL`
Topology discovery	LLDP	eth_type=0x88cc	Send to `CONTROLLER`
Controller IP resolution	ARP	arp, arp_tpa=c1 IP arp, arp_spa=c1 IP ... arp, arp_tpa=cN IP arp, arp_spa=cN IP	Send to `NORMAL`
Controller Self-Discovery	ARP	arp, arp_tpa=arbitrary IP	Send to `CONTROLLER`
C2C synchronization	TCP	ip, ip_src=c1 IP, ip_dst=c2 IP ip, ip_src=c2 IP, ip_dst=c1 IP ... ip, ip_src=cX IP, ip_dst=cY IP ip, ip_src=cY IP, ip_dst=cX IP	Send to `NORMAL`
Network Extension (NEXT): Dynamic switch IP address configuration	DHCP	in_port=`TREE` port, udp, udp_src=68 in_port=`TREE` port, udp, udp_src=67	Send to other `TREE` ports
NEXT: Remote switch configuration	SSH	in_port=`TREE` port, tcp, tcp_src=22 in_port=`TREE` port, tcp, tcp_dst=22	Send to other `TREE` ports
NEXT: C2S OF interaction	OF	in_port=`TREE` port, tcp, tcp_dst=6633 in_port=`TREE` port, tcp, tcp_src=6633	Send to other `TREE` ports
NEXT: Any ARP traffic	ARP	in_port=`TREE` port, arp	Send to other `TREE` ports
NEXT: Network extension discovery	DHCP	in_port=`INACTIVE` port, udp, udp_src=68	Send to `CONTROLLER`

Table 8.2: HSS - Initial and NEXT flow rules installed on switches throughout `Phase 2`.

8.4.2.2 `Phase 2b` - Enabling Control Plane Resilience

After the initial rules embedding, the leader controller disables (R)STP on each switch in order to fully discover the underlying topology. To ensure the topology is entirely discovered, the leader controller waits for a predefined period (ref. Section 8.6.2) and eventually computes and installs the resilient rules. Resilient control flows are installed for each S2C and C2C pair as per Section 8.4.4. Finally, the controller removes the initial `Phase 2a` rules (except for LLDP and ARP).



(a) S2C resilient paths (b) C2C resilient paths

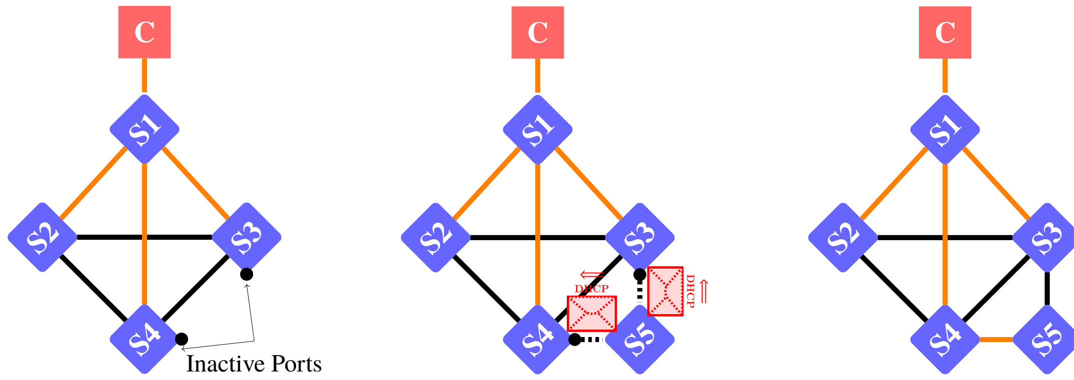
Figure 8.2: Exemplary resilient path output for $k = 1$ for: (a) S2C paths between `S4` and all controllers; and (b) all C2C pairs.

After finalizing `Phase 2b`, the control plane is resilient according to Section 8.1 requirements. Data plane failures are covered using the disjoint paths. Leader controller failures are covered by deploying multiple backup replicas.

8.4.3 `Phase 3`: Dynamic Network Extensions

To enable dynamic network extensions at runtime, the managed switches which are direct neighbors of the newly booted switches must forward the control plane traffic between the newly connected switches and controllers. In particular, DHCP, SSH, OF and ARP traffic forwarding must be enabled so that newly added switches can be bootstrapped. Since HSS disables (R)STP prior to executing `Phase 2b`, the rules that should match above mentioned traffic may not rely on MAC-learning based `NORMAL` forwarding, due to potential broadcast storms. Instead, the leader controller maintains a virtual tree topology, used to compute and install the NEXT rules that broadcast the control plane traffic to and from newly attached switches on the tree links only. Due to generic match semantics, NEXT rules are installed with a lower priority than `Phase 2b` rules.

A special discovery rule matches packets arriving on *inactive* ports (i.e., the last rule of Table 8.2). An inactive port is a port without an active neighbor, i.e., initially unconnected to another switch. If a new switch is connected to an already bootstrapped network, the discovery rule will forward its DHCP Discover message to controllers, encapsulated as a `PACKET-IN` message. The `PACKET-IN` contains the information about the ingress port the message arrived on, i.e., corresponding to the previously inactive port. The leader replicas processes the `PACKET-IN` by extracting the newly attached switch's MAC address, and adding the previously inactive port to the existing tree. If the new switch is connected to the existing network with multiple links, only the first activated port is used. Fig. 8.3 illustrates this scenario.



(a) Initial virtual spanning tree. (b) Addition of a switch with 2 links. (c) Resulting virtual spanning tree.

Figure 8.3: Exemplary network extension with one switch and two links. c) depicts the resulting tree.

8.4.4 Control / Data Plane Failure Handling

Impact of Data Plane Failures on Control Plane Flows: To tolerate switch node / link failures, `Phase 2b` identifies $k+1$ resilient paths for each C2C and C2S connection pair, necessary to tolerate k arbitrary data plane failures. To this end, the leader controller executes $k+1$ subsequent Dijkstra runs per connection pair. After each run, the link metrics of the found path are multiplied by a factor of 1000. Thus, through this iterative metric adaptation, $k+1$ maximally disjoint paths are effectively computed (as in 45.3.3 of 802.1Q-2018 [216]). The leader controller then maps and installs these paths. Exemplary resilient paths for $k = 1$ are depicted in Fig. 8.2.

Resilient flow rules are created for ARP, OF and SSH flows. OF and SSH use TCP at transport layer, that takes care of duplicate packet elimination. Duplicate ARP requests and replies, on the other hand, are delivered duplicated to sink nodes, which does not negatively impact the correctness. Thus, with the "seamless" replication mechanism, individual data plane failures never lead to packet loss, as long as disjoint alternative paths exist. In [135, 136], the resulting global C2C traffic generated by topology state exchange was shown to grow quadratically with the number of controllers and linearly with the number of network elements. For moderate control plane sizes (3-9 controllers), the imposed control plane load was determined close to negligible - $\sim$ 6.7 Mbps of per-replica load in a 5-replica cluster [2]. Hence, we propose a form of conditional traffic policing for misbehaving redundant S2C and C2C flows but do not investigate this issue further.

Impact of Data Plane Failures on NEXT Tree: Failures of individual data plane elements must result in adaptation of the NEXT tree. The approach we followed in Section 8.4.3 assumes initial tree computation and embedding, following by the controllers computing an alternative tree for each possible data plane failure in the previously embedded tree. When a link in the underlying topology fails that is mapped to the currently active tree, the leader controller refreshes the rules with those of an alternative tree, proactively computed for that particular link / node change. During the transition period, no new switches can be admitted, but since data plane failures are rare events, we consider the additional delay in network extension, incurred by reactive tree re-embedding, a negligible disadvantage.

Handling Control Plane Failures: If the leader replica fails, the remaining controllers elect a new leader using the distributed leader election procedure, thus incurring a short interruption period where no client requests relying on consensus can be handled by the controllers [3]. If a follower replica fails, current leader and the operation of the system remain unaffected. The resilient control plane remains operational as long as the controller majority remains active.

Note: Tolerating transitional faults during Phases 1-2 is currently unsupported and is considered as future work.

8.5 HHS - Hop-by-Hop Iterative Automated Distributed Control Plane Bootstrapping

The Hop-by-Hop Scheme (HHS) model realizes an iterative approach to switch discovery. In contrast to HSS, it alleviates the need for (R)STP and thus the (R)STP expiration timer (ref. Section 8.6.2). As before, a number of minimum constraints must hold at network startup: i) Controllers are aware of IP addresses of other participants, or are capable of discovering them using standardized DNS queries; ii) Switches are initialized in *Secure* mode *without* (R)STP and with *disabled* In-band mode; iii) Switches are provisioned with controllers' public certificates (PKI) or symmetric MAC keys [59, 65].

Fig. 8.4 depicts the abstract sequence diagram of HHS. In Secure mode, a switch relies on the initial *generic* (non-customized) flow table rules, available at boot time. By leveraging these, in `Phase 1a` and `Phase 1b`, the controllers establish bilateral connections. In `Phase 1c`, switches are assigned management IP addresses and controller lists. Using the *generic* rules, the switches adjacent to controller establish their OF sessions. Appropriately in `Phase 2a`, the leader rolls out the control plane flow rules to these switches. The provisioned rules realize the spanning tree forwarding functionality, used for iterative propagation of next hop switch's control traffic to the controller. With each newly discovered network element, HHS iteratively updates the tree. In `Phase 2b`, the leader computes and installs resilient paths for all control plane flows, whenever such paths become feasible. The attachment of new switches to an already bootstrapped network is possible by gradually expanding the spanning tree.

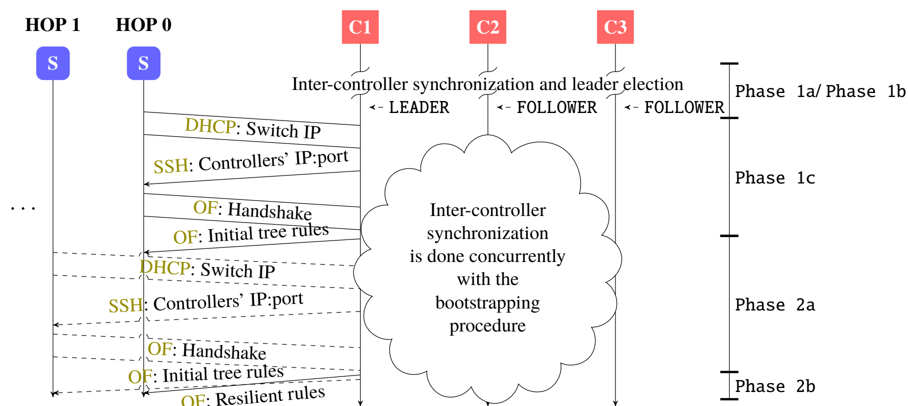


Figure 8.4: HHS - Message sequence diagram of the bootstrapping procedure as described in Section 8.5.

8.5.1 Phase 1: Network Startup and Initial Address Distribution

8.5.1.1 `Phase 1a` - Preconfigured Flow Rules

HHS assumes a set of initially preconfigured generic OF rules, necessary: i) to allow an initial connection with the controllers while in Secure mode; ii) to prevent broadcast storms in non-bootstrapped parts of a network.

In contrast to HSS, which bootstraps individual switches concurrently in FCFS manner, HHS bootstraps the network iteratively hop-by-hop, starting from switches adjacent to the leader controller. The non-customized *generic* rules (ref. Table 8.3) allow receiving traffic addressed to the switch itself, i.e., dropping any other traffic, except for the traffic generated by the switch itself. Traffic initiated by a switch is flooded on all its ports. This traffic should only be allowed to reach the leader controller, and is therefore, dropped by any neighboring switches. These rules thus prevent the occurrence of broadcast storms (special case being the C2C flow rules, ref. Section 8.5.1.2). Furthermore, they allow the controller to configure the switches connected directly to it.

Purpose	Packet Type	Matching	Action
Dynamic switch IP address configuration	DHCP	in_port=LOCAL, eth_src=switch_mac, udp, udp_src=68 udp, udp_src=67	Send to `ALL` Send to `LOCAL`
Remote switch configuration	SSH	in_port=LOCAL, eth_src=switch_mac, tcp, tcp_src=22 eth_dst=switch_mac, tcp, tcp_dst=22	Send to `ALL` Send to `LOCAL`
C2S OF interaction	OF	in_port=LOCAL, eth_src=switch_mac, tcp, tcp_dst=6633 eth_dst=switch_mac, tcp, tcp_src=6633	Send to `ALL` Send to `LOCAL`
S2C IP Resolution	ARP	in_port=LOCAL, eth_src=switch_mac, arp, arp_op=1 eth_dst=switch_mac, arp, arp_op=2	Send to `ALL` Send to `LOCAL`
C2S IP Resolution	ARP	in_port=LOCAL, eth_src=switch_mac, arp, arp_op=2	Send to `ALL`
C2S / Controller IP Resolution	ARP	arp, arp_op=1 arp, arp_op=2	Send to `ALL`
C2C Synchronization	TCP	tcp, tcp_src=2550 tcp, tcp_dst=2550	Send to `NORMAL`

Table 8.3: HHS - Preconfigured OF rules.

8.5.1.2 `Phase 1b` - Controller Synchronization

Excluding (R)STP implies we should avoid forwarding C2C synchronization traffic using `NORMAL` port so to avoid broadcast storms. However, Secure mode implies that this traffic must be handled with additional initial preconfigured rules that match and forward this traffic type (ref. last four rules of Table 8.3), prior to establishing OF connection with controller. Thus, it is impossible to come up with the generic set of preconfigured flow rules that do not leverage the `ALL` or `NORMAL` ports. Using either, however, initially results in broadcast storms. Therefore, apart from the preconfigured flow rules, we deploy a mechanism to cope with broadcast storms for C2C traffic (ref. Section 8.6.6).

8.5.1.3 `Phase 1c` - Distribution of Switch and Controller Connection Identifiers

The leader controller assigns the IP addresses initially only to its direct neighbor switches, and subsequently provisions them with controller lists. In order for the switches located two hops away from leader to receive their IP addresses, the generic preconfigured rules in the direct neighbor switches must first be extended with a new set of rules in `Phase 2a`.

8.5.2 Phase 2 - Enabling a Functional and Resilient Control Plane

In contrast to HSS, which computes the resilient control plane flows after disabling (R)STP, HHS tries to compute and deploy resilient flow rules whenever feasible. Namely, it installs the resilient flow rules as soon as there exist $k + 1$ disjoint paths for a single S2C communication pair. If the current discovered topology does not allow for identifying all required paths, flow rules are provisioned for a single path only. Whenever there is a change in topology, the leader retries computing the remaining disjoint paths.

8.5.2.1 Phase 2a - Initial OpenFlow Flow Rules

In this sub-step, leader provides the direct neighbor switches with rules that allow for the next-hop switches to communicate with all controllers (ref. Table 8.4). These rules have a lower priority than the resilient rules computed in `Phase 2b`. Since (R)STP is unavailable, broadcast storms must be avoided. Thus, in addition to the base topology discovered by LLDP- and ARP-probing, HHS maintains a virtual spanning tree.

The tree topology is updated and enforced upon switches on every topology change using Table 8.4 rules. Packets used in topology and controller discovery are sent directly to `CONTROLLER` port as `PACKET-IN`s. In OVS, packets forwarded to `CONTROLLER` port leverage the `NORMAL` data-path, which initially may seem problematic. However, due to not relying on Standalone operation, MAC-learning tables are empty and every packet is flooded instead. The flooded traffic cannot create broadcast storms as it can only reach two types of switches: i) those with `TREE` rules installed and, ii) those with generic preconfigured rules, which drop all traffic except their own. The discovery traffic is hence broadcasted only in the tree, since the `PACKET-IN`s match the OF type.

Purpose	Packet Type	Matching	Action
(NEXT) Dynamic switch IP address configuration	DHCP	in_port=TREE port, udp, udp_src=67 in_port=TREE port, udp, udp_src=68	Send to other TREE ports
(NEXT) Remote switch configuration	SSH	in_port=TREE port, tcp, tcp_dst=22 in_port=TREE port, tcp, tcp_src=22	Send to other TREE ports
(NEXT) C2S OF interaction	OF	in_port=TREE port, tcp, tcp_src=6633 in_port=TREE port, tcp, tcp_dst=6633	Send to other TREE ports
(NEXT) Any ARP traffic	ARP	in_port=TREE port, arp	Send to other TREE ports
Topology Discovery	LLDP	eth_type=0x88cc	Send to CONTROLLER
Controller self-discovery	ARP	arp, arp_tpa=arbitrary IP	Send to CONTROLLER
(NEXT) Network extension discovery	DHCP	in_port=INACTIVE port, udp, udp_src=68	Send to CONTROLLER

Table 8.4: HHS - Initial and NEXT flow rules installed on switches in `Phase 2a`.

8.5.3 Phase 2b: Enabling a Resilient Control Plane

To compute resilient paths, same as in Section 8.4.2.2, we deploy Dijkstra's algorithm. HHS does not assume visibility of entire topology to compute per-switch resilient paths. Instead, resilient paths are installed iteratively, whenever disjoint paths become available. This results in a quicker control plane resilience, as confirmed by the experimental results in Section 8.7.

8.5.4 Phase 3: Dynamic Network Extensions

To support dynamic network extensions, HHS adopts the same idea as HSS. Main difference to HSS is that HHS enforces the virtual tree topology together with remaining initial flows, i.e., the spanning tree

is (re-)enforced iteratively during the bootstrapping procedure itself. Compared to the rules installed in `Phase 2`, a single discovery rule per inactive switch port must be additionally installed, in order for new switches to successfully register with controllers (ref. last rule of Table 8.4). Whenever an element of the tree fails, HHS refreshes the rules with the alternative tree (ref. Section 8.4.4).

8.6 Selected Design and Evaluation Aspects

We next discuss selected design and evaluation aspects of the two automated bootstrapping schemes.

8.6.1 Flow Table Occupancy

Both bootstrapping schemes enforce a non-negligible number of forwarding rules. The exact Flow Table Occupancy (FTO) can be pre-determined only for loop-free topologies. In non-loop-free topologies, the FTO varies depending on the outputs of the used tree computation and routing algorithm. However, the lower and the upper bound FTO can always be calculated from derived expressions. For brevity, we next provide only the upper FTO bounds for both schemes.

HSS: An HSS-bootstrapped switch has its FTO upper-bounded by F_{HSS} rules:

$$\begin{gathered} F_{HSS} \leq n \cdot 4 + (5 + 3 \cdot |C|) + i \cdot 7 + m \cdot j \cdot 6 + k \\ n \in [0, \tbinom{|C|}{2}]; i \in [1, D_{Tree}]; m \in [0, |C|]; j \in [0, |S| - 1]; k \in \mathbb{N} \end{gathered} \tag{8.1}$$

where $|C|$ denotes the number of deployed controllers, D_{Tree} is the maximum node degree of the computed spanning tree, and $|S|$ is the number of switches.

The 4 fixed rules of Table 8.2 are TCP and ARP rules that allow for controller synchronization. The 5 fixed rules are composed of: ARP, SSH, OF rules for forwarding incoming traffic from controllers to the `LOCAL` port; and 2 discovery rules (LLDP, ARP). 3 flow rules (ARP, SSH, OF) are embedded per controller so to forward local traffic toward the respective controller (ref. Table 8.2). Index i denotes the degree of a switch in the virtual spanning tree used for network extensions. The 7 fixed rules are the NEXT discovery rules. Index j denotes how many resilient paths that start / end in a particular controller traverse a switch. Index m denotes the number of controller replicas. The 6 fixed rules are the resilient flow rules (ARP, SSH, OF) used for traffic relaying, in directions to / from other switches. k denotes the number of inactive ports, imposing a discovery rule per port.

HHS: HHS's FTO is upper-bounded by F_{HHS}:

$$F_{HHS} \leq 13 + n \cdot 4 + (5 + 3 \cdot |C|) + i \cdot 7 + m \cdot j \cdot 6 + k \tag{8.2}$$

HHS's flow table has the same composition as HSS, except for the additional 13 preconfigured rules (ref. Table 8.3).

8.6.2 (R)STP Timer Parametrization

HSS assumes that in `Phase 2a` all available switches are discovered and are provided with necessary flow rules. Hence, in `Phase 2b` the leader controller proceeds to disable (R)STP on all switches, so to

enable the blocked ports and populate the network topology view. Safe expiration timer T_{RSTP} after which to disable (R)STP can be estimated as:

$$T_{RSTP} \geq T^{R}_{DHCP} + T^{HS}_{DHCP} + T_{C_{list}} + T^{HS}_{OF} + T^{FO}_{RSTP}$$

where T^{R}_{DHCP} represents the transmission interval of DHCP Discover packets by the switches' DHCP clients; T^{HS}_{DHCP} is the delay imposed by a successful DHCP handshake; $T_{C_{list}}$ is the time required to deliver the controllers' IP address list; T^{HS}_{OF} comprises the OF session establishment time and time required to install and confirm the initial flows (ref. `Phase 2a`); and T^{FO}_{RSTP} is the time required for (R)STP to recover from potential network failures during bootstrapping. Worst-case time necessary to recover the spanning tree after switch / link failures with Spanning Tree Protocol (STP) may reach up to $50s$ [209]. In corner cases, the recovery time for Rapid Spanning Tree Protocol (RSTP) may increase up to $120s$ (ref. Count-to-Infinity problem [66]). In experiments conducted in [66], however, the worst-case (R)STP recovery time in a 16-switch topology peaked at $50s$. If a new switch sends a DHCP Discover message before the timer expiration, HSS preempts the timer. On a successful expiration, HSS disables (R)STP on the switches as per `Phase 2b`. Parameters used in evaluation were deduced empirically (ref. Table 8.5).

Parameter	T^{R}_{DHCP}	T^{HS}_{DHCP}	$T_{C_{list}}$	T^{HS}_{OF}	T^{FO}_{RSTP}
Value	$1s$	$1s$	$1s$	$2s$	$50s$ [66]

Table 8.5: Parametrization of the (R)STP timer in HSS.

8.6.3 Network Topology Discovery

ODL's *OpenFlowPlugin* relies on LLDP flow rules to learn the network topology. The controllers periodically output OF `PACKET-OUT` messages with encapsulated LLDP Data Units (LLDPDUs) on each switch's port. Neighbor switches then receive and forward these packets back to their Master. Thus, controllers learn about the topology adjacency. By default, *OpenFlowPlugin* transmits the LLDPDUs each $5s$. If no probes are received for 3 consecutive periods, the link originating in that port is considered unavailable and is removed from the data store. To facilitate faster discovery of switches, we increase the rate of LLDPDU transmissions to $1s$. Since the additional control plane load is generated only on unused ports, this optimization imposes no overhead.

8.6.4 Controller Placement Discovery

ODL's *hosttracker* discovers hosts and populates the data store after observing ARP, IPv4, or Internet Protocol v6 (IPv6) frames on previously inactive ports. To discover controllers' placement relative to topology, we leverage a self-discovery rule (ref. Table 8.2). Controllers periodically send self-discovery probes into the network, the edge switches subsequently encapsulate the ARP probes as `PACKET-IN`s and forward them to their Master controller. The *hosttracker* next processes the `PACKET-IN`, matches the probe's source IP with a host and tags that host as an active controller.

8.6.5 Overhead of Controller Clustering

On a newly discovered switch, the *OpenFlowPlugin* module of an ODL replica attempts to take its ownership by initiating a role request to become its Master. Another controller replica may, however, be elected as the Raft leader of the node inventory data store used to serialize (i.e., reach consensus on and order) switches' state modifications. The election of the OF Master is thus independent of the inventory data store ownership and the leader of the bootstrapping procedure. According to the OF specification, only the Master of a switch may directly modify its flow table. For this reason, multiple controllers may exchange data in order to apply changes to a switch's flow table. Fig. 8.5 illustrates the worst case relevant for our evaluation.

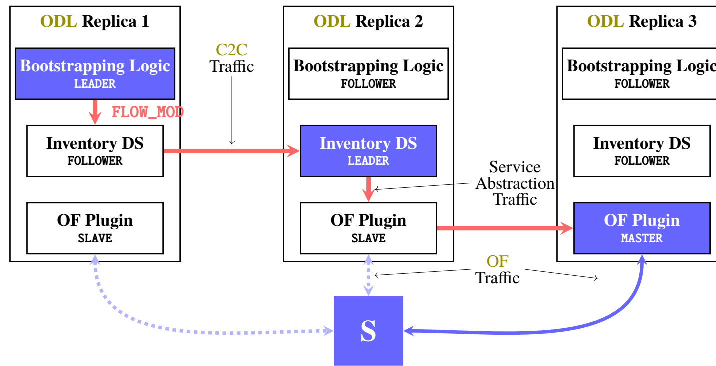


Figure 8.5: An exemplary data flow of a FLOW_MOD RPC with each entity's leader on a different controller.

8.6.6 Coping With Broadcast Storms in HHS

To solve the issues related to `Phase 2b` where particular flows may initially cause broadcast storms, we rely on rate limiting mechanisms provided by the data plane (e.g., OF's Metering [122] or Linux's Traffic Control). Namely, we police the following C2C flows (ref. Table 8.3): i) controller-initiated ARP traffic; ii) TCP Synchronize Messages (TCP SYNs) destined for the C2C TCP port; ii) TCP SYN Acknowledgments (ACKs) with C2C TCP port as source. The rate limit for policers may be configured to a very low value, e.g., we used $1.5kbps$ for metering both ARP requests and replies ($\sim$ 12 ARP pps). Similarly, a low maximum rate can be chosen for TCP SYNs and TCP SYN ACKs. It suffices to match and rate-limit only TCP SYN and TCP SYN ACK traffic (as only these packets may generate broadcasts) and not the complete TCP flow.

8.7 Evaluation and Results

8.7.1 Evaluation Methodology

We have implemented HSS and HHS as extensions of ODL's `Boron-SR3` release and have evaluated their performance against realistic emulated industrial topologies [52, 215, 85, 192]. We focus on the impact of topology type on the bootstrapping efficiency and not on the impact of network size. This said, we did successfully validate our approaches in topologies comprising up to $\sim$ 64 switches in a

single L2 domain, which are larger than the average industrial topologies with requirements on strict QoS [85, 192].

Our network emulator generates the input topology by isolating OVS instances in Docker containers and interconnecting them as per target topology. In the single-controller scenario, the controller and the topology were hosted on PC equipped with a recent Intel Core i7 CPU. Multi-controller scenarios were executed on a dual-CPU Intel Xeon E5 server.

The schemes are compared along with the following KPIs:

Global Bootstrapping Convergence Time (GBCT): Defined as the difference between: i) the time instant at which all switches are provisioned with resilient flow rules; and ii) the instant when the first switch was observed by any controller.

Time To Extend (TEXT): For each added switch, we measure and average the difference between the instants: i) when a switch is provided with the control flow rules; ii) when the switch was first observed by a controller.

Flow Table Occupancy (FTO): Number of active flow entries in a flow table after successful bootstrapping procedure.

Industrial topologies, in particular those often found in process industry and factory automation, were selected due to their requirement on redundant communication and dynamic network adaptation [204]. Fig. 8.6 depicts the evaluated topologies, with corresponding suffixes denoting the topology size. *line-N* and *star-N* topologies were evaluated with a single controller only, while the remaining topologies deploy up to 3 controllers. Note that while *line-N*, *star-N* and *1-ring-N* topologies do not satisfy the conditions for path disjointness, we also evaluate these topologies to gain additional insights. In single-controller scenarios, the replica was always placed adjacent to S_1. For scenarios involving 3 controllers, controllers were placed according to Table 8.6. In general, the controller placement influences the observed measured KPIs in both single- and multi-controller scenarios. For brevity, we limit our discussion to the impact of the Raft leader placement.

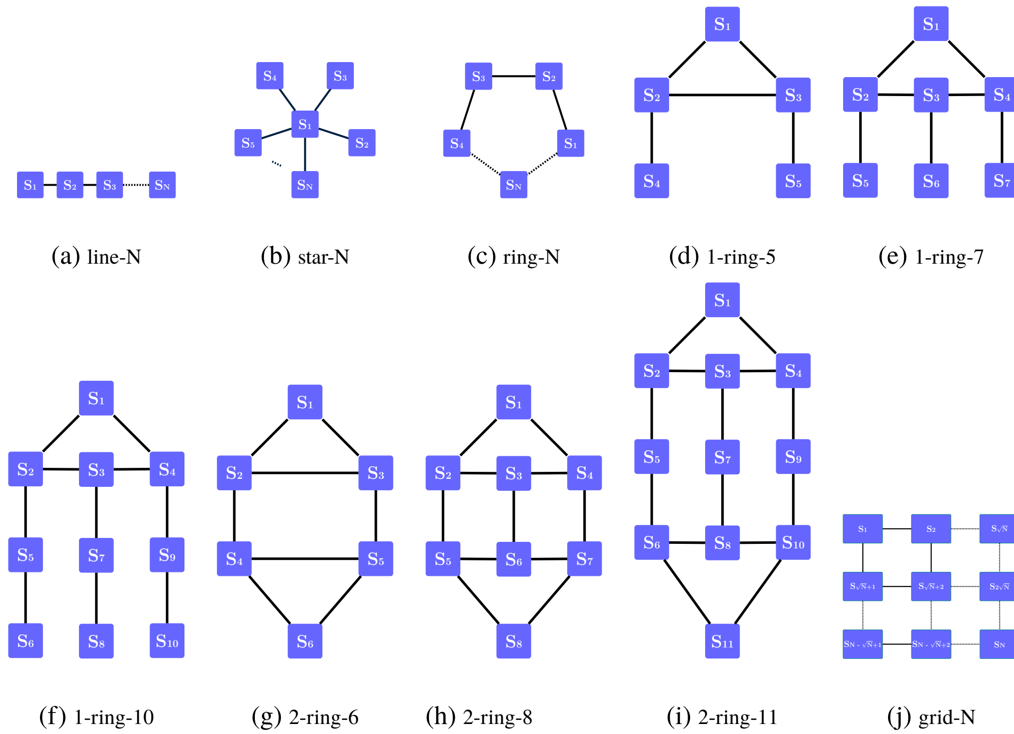
Topology	**Controller placement**	**Topology**	**Controller placement**
ring-4	S_1, S_2, S_3	1-ring-5	S_1, S_4, S_5
ring-8	S_1, S_4, S_6	1-ring-7	S_1, S_3, S_7
ring-16	S_1, S_6, S_{11}	1-ring-10	S_1, S_3, S_{10}
grid-4	S_1, S_2, S_3	2-ring-6	S_1, S_3, S_6
grid-9	S_1, S_5, S_9	2-ring-11	S_1, S_7, S_{11}

Table 8.6: Controllers' placement with 3 controllers.

8.7.2 Bootstrapping Convergence Time

8.7.2.1 Single-Controller Setup

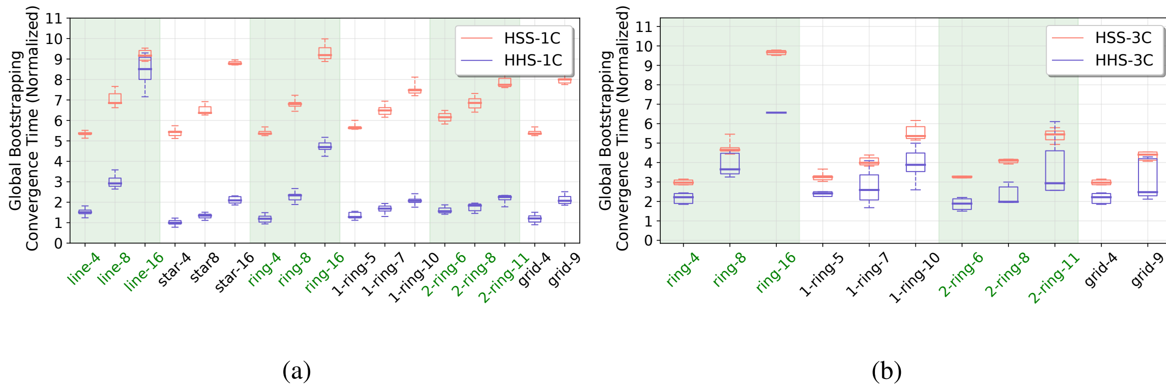
GBCTs are depicted in Fig. 8.7. GBCT of HSS is not impacted by the type of the underlying topology. Instead, the time to embed resilient paths increases with the overall topology size. This is due to the fact that HSS's `Phase 1c` and `Phase 2` do not execute in a concurrent manner. HHS's GBCT, on the other hand, is dependent on the design of the underlying topology. In particular, its convergence time



(a) line-N (b) star-N (c) ring-N (d) 1-ring-5 (e) 1-ring-7

(f) 1-ring-10 (g) 2-ring-6 (h) 2-ring-8 (i) 2-ring-11 (j) grid-N

Figure 8.6: Considered industrial topologies in evaluation of HSS and HHS [52, 215].



(a) (b)

Figure 8.7: Observed GBCT for single- and 3-controller scenarios. Measured time is normalized with respect to the minimum observed GBCT means (13.5s and 33.9s for and 1 and 3 controllers, respectively). HHS outperforms HSS for all evaluated topologies, mostly due to HSS being lower bounded by the (R)STP timer (ref. Section 8.6.2).

scales with the number of additional hops the control traffic must traverse to reach a certain switch in the network.

HHS bootstraps the switches with same hop distance from the controller concurrently. For example, in grid topologies, the maximum number of hops between a switch and controller does not increase with the same rate as the number of switches. Thus, going from *grid-4* to *grid-9*, the absolute number of switches increases by 5, but the hops required for HHS's controller to reach most distant switches increases only by 2, leading to an increased performance gap over HSS. Additionally, in contrast to

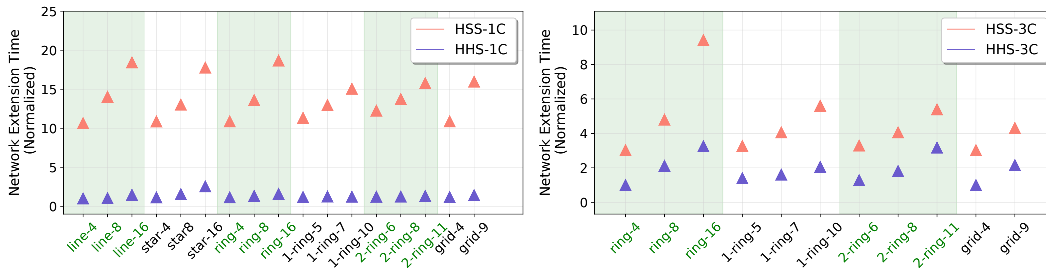


Figure 8.8: TEXT values of the two schemes for configurations deploying 1 to 3 controllers. Y-axis depicts the (per-topology) normalized TEXT, relative to the lowest obtained mean TEXT, i.e., 6.5*s* and 33.5*s* for 1- and 3-controllers.

HSS, HHS does not suffer from an artificially introduced lower bound (due to its non-reliance on (R)STP).

8.7.2.2 Multi-Controller Setup

HHS relies heavily on the controllers' data store during its operation - each read / write operation requires proxying the request through the current Raft leader. Additionally, controllers frequently block and synchronize their data stores, thus adding additional processing latency in sequential code execution. In contrast to HHS, HSS does not rely on the data store as much, providing for better scaling with increasing control plane size, hence the drop in performance gap for the 3-controller scenario.

With HSS, the Raft leader's placement does not impact the resulting performance, since the switches are bootstrapped in the FCFS manner. Thus, HSS's values in Fig. 8.7b are generally less spread than for HHS. For example, with HHS *grid-9* may be bootstrapped as quick as *grid-4*. If the leader for *grid-9* is elected adjacent to S_5 (ref. Fig. 8.6), the leader will require 2 hops to reach any switch, i.e., the same number of hops as required by *grid-4* for a leader placed on any of the 4 switches. Note, however, that such comparison does not hold in ring topologies, since the leader placement there does not influence the maximum hop distance to leader.

8.7.2.3 Discussion

In general, HHS outperforms HSS for all evaluated topologies. This is due to the lower bound on the GBCT in HSS as imposed by the (R)STP timer (set to 55*s*, ref. Section 8.6.2). In HSS, GBCT increases linearly with the topology size, contrary to HHS, where GBCT exhibits a non-linear relation with the maximal hop distance between the leader controller and the switch. The larger the distance, the larger the increase in necessary bootstrapping time. Thus, the performance gap between HSS and HHS depends on the placement of the Raft leader and the overall topology size. In 3-controller scenarios, HHS's extensive reliance on the distributed data store reduces the performance gap.

8.7.3 Network Extension Time

8.7.3.1 Single-Controller Setup

In single-controller scenarios, HHS requires a nearly constant time to extend a topology, i.e., deploy a new switch, independent of the existing topology and number of new switches (ref. Fig. 8.8). The slight increase for larger topology sizes relates to the accumulated CPU load from having to consider additional switches, e.g., additional LLDP packets to process when refreshing the topology and spanning tree computation overhead. For HSS, just like with GBCT, TEXT grows linearly with the topology size. This is due to the waiting period related to disabling the (R)STP timer and contention in sequential rule installation. An optimistic case is the single-switch extension where no contention impacts rule installation order (not depicted).

8.7.3.2 Multi-Controller Setup

A newly added switch in a multi-controller setup must on average wait longer on its control flow rules. The C2C synchronization results in a higher TEXT degradation for HHS than for HSS. Additionally, compared to the single-controller case, where TEXT remains mostly constant, in scenarios with 3 controllers HHS's TEXT grows linearly with topology size, due to a larger resulting C2C separation.

8.7.3.3 Discussion

HHS outperforms HSS both in single- and 3-controller scenarios. However, due to the distributed synchronization overhead, and the fact that control flow rules can be provisioned only when a controller is discovered, the performance gap between HSS and HHS is reduced.

8.7.4 Flow Table Occupancy

Fig. 8.9 portrays the substantial growth in the FTO when deploying 3 controllers instead of a single one. This is is due to switches being provisioned with resilient flow rules for connections to all controllers. Additionally, some switches contain rules used to forward the inter-controller traffic. However, the change in FTO is influenced not only by number of controllers, but also by the topology size, degree of connectivity, *and* the controller placement. The ratios of FTOs are summarized in Table 8.7.

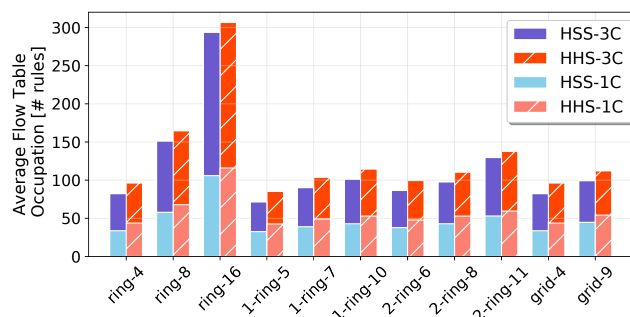


Figure 8.9: Bar charts comparing the average FTO for HSS and HHS bootstrapping schemes.

In HSS, the placement of the leader does not influence the FTO. This is due to resilient and tree rules being installed only *after* a successful discovery of the entire network. Thus, the FTO depends only on the output of the routing and tree computation algorithm. On the contrary, in HHS the placement of the Raft leader controller influences the output of the iteratively-built spanning tree, producing fluctuating FTOs for repeated executions. Notably, in *grid-N*, *x-ring-N* topologies, the leader controller placement influences the FTO fluctuations, while in *ring-N* topologies, different leader placements have no effect on FTO (visual results omitted due to space considerations). In all evaluated scenarios, HSS results in lower minimal, average, and maximal FTOs. The average difference in FTO between HSS and HHS equals approximately the number of preconfigured rules in HHS scheme. Indeed, both bootstrapping schemes enforce a non-negligible number of forwarding rules. Investigation of methods for shrinking the number of active flow rules leveraged by the schemes, i.e., by means of flow table compression [37, 159] should be considered in future studies.

Topology	HSS	HHS
ring-{4, 8, 16}	{2.44, 2.62, 2.77}	{2.2, 2.43, 2.43}
grid-{4, 9}	{2.44, 2.22}	{2.2, 2.07}
1-ring-{5, 7, 10}	{2.19, 2.31, 2.36}	{2.0, 2.12, 2.16}
2-ring-{6, 8, 11}	{2.29, 2.27, 2.44}	{2.08, 2.07, 2.34}

Table 8.7: Ratios of observed average per-switch FTOs. Values are normalized respective to the FTO in 1-controller case for the same scheme and topology.

8.8 Chapter Summary and Outlook

We have described the designs of the first two bootstrapping schemes that autonomously bootstrap a multi-controller SDN with a resilient control plane and with an automated IP and controller list provisioning to switches. Besides evidencing the practicability of these two approaches and quantifying the trade-offs they reveal (implementation complexity, legacy protocols needed, convergence time, flow table occupancy, network extension time) the proposed approaches open the door towards reliable SDN deployments in environments where OOBC connections are not possible, e.g., industrial networks. This in return enables the deployment of orthogonal advances in industrial SDN environments [74, 85, 114, 4]. To undermine the practicability of the presented schemes, the proposed automated bootstrapping was validated successfully in an operational industrial network with fail-safe requirements [9, 15] in the context of Horizon 2020 project VirtuWind [1].

Outlook: The proposed approaches can be further improved by a number of orthogonal additions:

- *Methods to alleviate traffic starvation*: Having the control and data plane share the same infrastructure motivates investigation of proper isolation of both traffic types and ensuring non-starvation of control traffic. This is especially relevant in industrial environments where data plane traffic often has stringent QoS requirements.
- *Scalability of the bootstrapping schemes*: Control plane load may eventually become a scalability bottleneck due to imposed C2S and C2C traffic. This load, however, scales linearly with

[1] VirtuWind - H2020-ICT-14-2014 - http://www.virtuwind.eu

the number of switches and is hence predictable and easy to provision the network resources for. Similarly, FTO may eventually become problematic due to multiplicative m (number of controllers) and j (number of resilient flows crossing the observed switch) terms in Eq. 8.1 and Eq. 8.2. To minimize the overhead of resulting FTO, the applicability of existing approaches for flow table compression and flow rule context switching [42] should be investigated for further optimization.

- *IPv6 support*: Supporting the bootstrapping of IPv6-enabled SDNs is a relevant extension currently considered for future work. Distribution of addresses with Dynamic Host Configuration Protocol Version 6 (DHCPv6) seems trivial - the DHCP server in the ODL instances would, however, require a corresponding upgrade. Propagation of Router Advertisements for purpose of enabling Stateless Address Autoconfiguration (SLAAC), would be non-trivial with HHS due to non-reliance on (R)STP and possible occurrence of broadcast storm in the initial phase (directly after network startup), due to use of `NORMAL` / `ALL` logical output port. Hence, similarly to other broadcasted traffic, it would require additional default flow rules in each switch and a configured policing mechanism for limitation of storm impact (ref. Section 8.6.6).

Chapter 9

Conclusion

9.1 Thesis Summary

This thesis postulates a working SDN control plane model where network services are replicated across replicas in a logical cluster, in order to tolerate Fail-Stop and Byzantine faults. We conclude that a higher control plane robustness, i.e., a higher degree of availability and reliability is achievable without an impactful setback in achievable correctness, response time and complexity of the resulting control plane deployment.

Consensus protocols ensure sequencing of client requests in an ordered log and its replication to the replicas of the logical cluster. Our work introduces a framework for analytic estimation of the availability and response time of a distributed control plane built atop of such consensus protocols. We have shown that a proper configuration and replication intensity of the control plane can alleviate the issue of SPOF with resulting worst-case unavailability $< 1e^{-9}$, suitable for even the most critical of industrial services. We have shown how, using a suitable consensus protocol parametrization and SDN application workload, the strict control plane response time requirements postulated by the future TSN-enabled applications can be provably met by a properly parametrized distributed control plane. We have also handled the corner cases related to correct consensus operation in the face of network partitions and have, in this light, proposed a modular design that decouples the consensus implementation from underlying failure detection.

To provide for a decreased response time in the average case, we have introduced and validated the AC state synchronization model, suitable for applications enabled to operate correctly with some degree of data state staleness (e.g., distributed load-balancers, best-effort traffic routing etc.). By not relying on costly consensus on each resource state update, the clients of the distributed control plane benefit from shortened request-handling time. The AC provides for guarantees on maximal state divergence across replicas and thus provides for provable worst-case inconsistency among replicas at all times.

We have also introduced MORPH, a control plane design that tolerates both Byzantine and Fail-Stop control plane failures. Compared to Fail-Stop-only tolerant designs, it efficiently enhances the reliability and availability of controller-based decision-making in industrial environments. MORPH's distinguishing property is that, following an exposure of a controller failure, it subsequently allows

for optimization of cluster membership according to type of failure detected. Together with its extension P4BFT, it autonomously adapts the cluster configuration to minimize the distributed control plane overhead at all times. P4BFT's control plane / data plane co-design additionally offloads the packet processing from general purpose CPUs to optimally selected data-plane NPUs, thus leading to a decrease in request processing time for controller state updates and switch reconfigurations. To minimize the response time overhead of BFT sequencing step, we have designed and evaluated three novel BFT sequencing protocols and have confirmed their response time decrease in realistic topologies.

To lower the entry-barrier in deployment of the presented control plane designs, we have proposed and published two bootstrapping schemes for automated bootstrapping of multi-controller SDNs. The proposed schemes target in-band control plane deployment with redundant control flow establishment for enhanced availability of C2S and C2C connections. The schemes were validated in a number of emulated network topologies typical for industrial deployments and in a production deployment of an industrial SDN operated by a utility provider [15].

9.2 Concluding Remarks

We expect the presented designs to enable future scalable, highly-resilient and distributed SDN control plane designs. While centralized TSN management may offer an initial deployment environment, generic service orchestration designs may directly benefit from the presented novelties as well. In particular, the dependability of deployment and orchestration tools operating on VNFs, unikernels and microservices, as well as the dependability of tools for wiring of hardware devices, APIs and IoT services (e.g., Node-RED) can benefit from the presented concepts. Finally, we strongly hope that the control plane analysis and presented designs pursued here, can lead to a reduction of the remaining stigma surrounding the reliability, scalability and applicability of logically centralized SDN control plane and instead pave the way for innovation in generally conservative and slowly adapting Industrial Ethernet networks.

Bibliography

Publications by the author

Journal publications

[1] **E. Sakic**, N. Đerić, and W. Kellerer. "MORPH: An Adaptive Framework for Efficient and Byzantine Fault-Tolerant SDN Control Plane." In: *IEEE Journal on Selected Areas in Communications* 36.10 (Oct. 2018), pp. 2158–2174. ISSN: 1558-0008. DOI: 10.1109/JSAC.2018.2869938.

[2] **E. Sakic** and W. Kellerer. "Impact of Adaptive Consistency on Distributed SDN Applications: An Empirical Study." In: *IEEE Journal on Selected Areas in Communications* 36.12 (Dec. 2018), pp. 2702–2715. ISSN: 1558-0008. DOI: 10.1109/JSAC.2018.2871309.

[3] **E. Sakic** and W. Kellerer. "Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane." In: *IEEE Transactions on Network and Service Management* 15.1 (Mar. 2018), pp. 304–318. ISSN: 2373-7379. DOI: 10.1109/TNSM.2017.2775061.

[4] P. Vizarreta, A. V. Bemten, **E. Sakic**, K. Abbasi, N. E. Petroulakis, W. Kellerer, and C. M. Machuca. "Incentives for a Softwarization of Wind Park Communication Networks." In: *IEEE Communications Magazine* 57.5 (May 2019), pp. 138–144. ISSN: 1558-1896. DOI: 10.1109/MCOM.2019.1800492.

Conference publications

[5] **E. Sakic**, M. Avdic, A. Van Bemten, and W. Kellerer. "Automated Bootstrapping of A Fault-Resilient In-Band Control Plane." In: *Proceedings of the Symposium on SDN Research*. SOSR '20. San Jose, CA, USA: Association for Computing Machinery, 2020, pp. 1–13. ISBN: 9781450371018. DOI: 10.1145/3373360.3380829. URL: https://doi.org/10.1145/3373360.3380829.

[6] **E. Sakic**, N. Deric, E. Goshi, and W. Kellerer. "P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane." In: *2019 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2019, pp. 1–7. DOI: 10.1109/GLOBECOM38437.2019.9013772.

[7] **E. Sakic** and W. Kellerer. "BFT Protocols for Heterogeneous Resource Allocations in Distributed SDN Control Plane." In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. May 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761956.

[8] **E. Sakic** and W. Kellerer. "Decoupling of Distributed Consensus, Failure Detection and Agreement in SDN Control Plane." In: *IFIP Networking 2020 Conference (IFIP Networking 2020)*. Paris, France, June 2020.

[9] **E. Sakic**, V. Kulkarni, V. Theodorou, A. Matsiuk, S. Kuenzer, N. E. Petroulakis, and K. Fysarakis. "VirtuWind – An SDN- and NFV-Based Architecture for Softwarized Industrial Networks." In: *Measurement, Modelling and Evaluation of Computing Systems*. Ed. by R. German, K.-S. Hielscher, and U. R. Krieger. Cham: Springer International Publishing, 2018, pp. 251–261. ISBN: 978-3-319-74947-1. DOI: 10.1007/978-3-319-74947-1_17.

[10] **E. Sakic**, F. Sardis, J. W. Guck, and W. Kellerer. "Towards adaptive state consistency in distributed SDN control plane." In: *2017 IEEE International Conference on Communications (ICC)*. May 2017, pp. 1–7. DOI: 10.1109/ICC.2017.7997164.

[11] **E. Sakic**, C. B. Serna, E. Goshi, N. Deric, and W. Kellerer. "P4BFT: A Demonstration of Hardware-Accelerated BFT in Fault-Tolerant Network Control Plane." In: *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. SIGCOMM Posters and Demos '19. Beijing, China: Association for Computing Machinery, 2019, pp. 6–8. ISBN: 9781450368865. DOI: 10.1145/3342280.3342289. URL: https://doi.org/10.1145/3342280.3342289.

[12] **E. Sakic**, P. Vizarreta, and W. Kellerer. "SEER: Performance-Aware Leader Election in Single-Leader Consensus." In: *[Submitted for publication]*. Mar. 2021.

[13] K. Fysarakis, N. E. Petroulakis, A. Roos, K. Abbasi, P. Vizarreta, G. Petropoulos, **E. Sakic**, G. Spanoudakis, and I. Askoxylakis. "A Reactive Security Framework for operational wind parks using Service Function Chaining." In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. July 2017, pp. 663–668. DOI: 10.1109/ISCC.2017.8024604.

[14] N. E. Petroulakis, E. Lakka, **E. Sakic**, V. Kulkarni, K. Fysarakis, et al. "SEMIoTICS Architectural Framework: End-to-end Security, Connectivity and Interoperability for Industrial IoT." In: *2019 Global IoT Summit (GIoTS)*. June 2019, pp. 1–6. DOI: 10.1109/GIOTS.2019.8766399.

[15] N. E. Petroulakis, T. Mahmoodi, V. Kulkarni, P. Vizarreta, A. Roos, K. Abbasi, X. V. and Spiros Spirou, A. Matsiuk, **E. Sakic**, and I. Askoxylakis. "VirtuWind: Virtual and Programmable Industrial Network Prototype Deployed in Operational Wind Park." In: *2016 European Conference on Networks and Communications (EuCNC)*. Athens, Greece, Apr. 2016, pp. 412–416.

[16] J. Seeger, A. Bröring, M. Pahl, and **E. Sakic**. "Rule-Based Translation of Application-Level QoS Constraints into SDN Configurations for the IoT." In: *2019 European Conference on Networks and Communications (EuCNC)*. June 2019, pp. 432–437. DOI: 10.1109/EuCNC.2019.8802018.

[17] V. Theodorou, K. V. Katsaros, A. Roos, **E. Sakic**, and V. Kulkarni. "Cross-Domain Network Slicing for Industrial Applications." In: *2018 European Conference on Networks and Communications (EuCNC)*. June 2018, pp. 209–213. DOI: 10.1109/EuCNC.2018.8443241.

[18] P. Vizarreta, **E. Sakic**, W. Kellerer, and C. M. Machuca. "Mining Software Repositories for Predictive Modelling of Defects in SDN Controller." In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Apr. 2019, pp. 80–88.

Granted Patents and Patent Applications

[19] **E. Sakic**. *EP3461081 - METHOD FOR OPERATING A COMMUNICATION NETWORK COMPRISING MULTIPLE COMMUNICATION DEVICES OF AN INDUSTRIAL AUTOMATION SYSTEM, CONTROL UNIT AND COMMUNICATION DEVICE*. European Patent Office, Mar. 2019.

[20] **E. Sakic**. *EP3501140 - METHOD FOR OPERATING AN INDUSTRIAL AUTOMATION SYSTEM COMMUNICATION NETWORK COMPRISING A PLURALITY OF COMMUNICATION DEVICES, AND CONTROL UNIT*. European Patent Office, June 2019.

[21] **E. Sakic**. *US20190245805 - METHOD FOR OPERATING AN INDUSTRIAL AUTOMATION SYSTEM COMMUNICATION NETWORK COMPRISING A PLURALITY OF COMMUNICATION DEVICES, AND CONTROL UNIT*. United States of America, Aug. 2019.

[0] **E. Sakic** and A. Zirkler. *CONTROL UNIT AND METHOD FOR OPERATING AN INDUSTRIAL AUTOMATION SYSTEM COMMUNICATION NETWORK COMPRISING A PLURALITY OF COMMUNICATION DEVICES*. US Patent 10,890,901. Jan. 2021.

[22] **E. Sakic** and A. Zirkler. *EP3435180 - METHOD FOR OPERATING A COMMUNICATION NETWORK COMPRISING MULTIPLE COMMUNICATION DEVICES OF AN INDUSTRIAL AUTOMATION SYSTEM AND CONTROL UNIT*. European Patent Office, Jan. 2019.

[0] **E., Sakic**. *METHOD FOR OPERATING AN INDUSTRIAL AUTOMATION SYSTEM COMMUNICATION NETWORK COMPRISING A PLURALITY OF COMMUNICATION DEVICES, AND CONTROL UNIT*. US Patent 10,848,439. United States of America, Nov. 2020.

[23] A. Gruner, A. M. Houyou, H. P. Huth, and **E. Sakic**. *US20190173779 - METHOD, SOFTWARE AGENT, NETWORKED DEVICE AND SDN CONTROLLER FOR SOFTWARE DEFINED NETWORKING A CYBER PHYSICAL NETWORK OF DIFFERENT TECHNICAL DOMAINS, IN PARTICULAR AN INDUSTRY AUTOMATION NETWORK*. United States of America, June 2019.

[24] A. Gruner, A. M. Houyou, J. Riedl, and **E. Sakic**. *EP3462673 - METHOD FOR OPERATING A COMMUNICATION NETWORK OF AN INDUSTRIAL AUTOMATION SYSTEM AND CONTROL DEVICE*. European Patent Office, Apr. 2019.

[25] A. M. Houyou, H. P. Huth, **E. Sakic**, and A. Gruner. *EP3485617 - METHOD, SOFTWARE AGENT, NETWORKED DEVICE AND SDN-CONTROLLER FOR SOFTWARE DEFINED NETWORKING A CYBER-PHYSICAL NETWORK OF DIFFERENT TECHNICAL DOMAINS, IN PARTICULAR AN INDUSTRY AUTOMATION NETWORK*. European Patent Office, May 2019.

[0] V. Kulkarni and **E. Sakic**. *COMPUTER SYSTEM AND METHOD FOR DYNAMICALLY ADAPTING A SOFTWARE-DEFINED NETWORK*. US Patent 10,652,100. May 2020.

[26] V. Kulkarni and **E. Sakic**. *EP3526931 - COMPUTER SYSTEM AND METHOD FOR DYNAMICALLY ADAPTING A SOFTWARE-DEFINED NETWORK*. European Patent Office, Aug. 2019.

[27] V. Kulkarni and **E. Sakic**. *US20190268237 - COMPUTER SYSTEM AND METHOD FOR DYNAMICALLY ADAPTING A SOFTWARE-DEFINED NETWORK*. United States of America, June 2019.

[28] J. Riedl, **E. Sakic**, M. Zafirovic-Vukotic, E. Sakic, and A. Zirkler. *EP3499798 - METHOD FOR TRANSMITTING DATA WITHIN AN INDUSTRIAL COMMUNICATION NETWORK AND NETWORK CONTROLLER*. European Patent Office, June 2019.

General publications

[29] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers. “Fast Recovery in Software-Defined Networks.” In: *2014 Third European Workshop on Software Defined Networks*. Sept. 2014, pp. 61–66. DOI: 10.1109/EWSDN.2014.13.

[30] F. Ansah, M. A. Abid, and H. de Meer. “Schedulability Analysis and GCL Computation for Time-Sensitive Networks.” In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. Vol. 1. July 2019, pp. 926–932. DOI: 10.1109/INDIN41052.2019.8971965.

[31] F. Ansah, M. Majumder, H. de Meer, and J. Jasperneite. “Network Slicing : An Industry Perspective.” In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2019, pp. 1367–1370. DOI: 10.1109/ETFA.2019.8869073.

[32] M. Aslan and A. Matrawy. “A Clustering-based Consistency Adaptation Strategy for Distributed SDN Controllers.” In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. June 2018, pp. 441–448. DOI: 10.1109/NETSOFT.2018.8460120.

[33] M. Aslan and A. Matrawy. “Adaptive consistency for distributed SDN controllers.” In: *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. Sept. 2016, pp. 150–157. DOI: 10.1109/NETWKS.2016.7751168.

[34] A. Avritzer, L. Carnevali, H. Ghasemieh, L. Happe, B. R. Haverkort, A. Koziolek, D. Menasche, A. Remke, S. S. Sarvestani, and E. Vicario. “Survivability Evaluation of Gas, Water and Electricity Infrastructures.” In: *Electronic Notes in Theoretical Computer Science* 310 (2015). Proceedings of the Seventh International Workshop on the Practical Application of Stochastic Modelling (PASM), pp. 5–25. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2014.12.010. URL: http://www.sciencedirect.com/science/article/pii/S1571066114000942.

[35] V. Balegas, D. Serra, S. Duarte, C. Ferreira, M. Shapiro, R. Rodrigues, and N. Preguiça. "Extending Eventually Consistent Cloud Databases for Enforcing Numeric Invariants." In: *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. Sept. 2015, pp. 31–36. DOI: 10.1109/SRDS.2015.32.

[36] V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, N. Preguiça, M. Najafzadeh, and M. Shapiro. "Putting Consistency Back into Eventual Consistency." In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. Bordeaux, France: Association for Computing Machinery, 2015. ISBN: 9781450332385. DOI: 10.1145/2741948.2741972. URL: https://doi.org/10.1145/2741948.2741972.

[37] S. Banerjee and K. Kannan. "Tag-In-Tag: Efficient flow table management in SDN switches." In: *10th International Conference on Network and Service Management (CNSM) and Workshop*. Nov. 2014, pp. 109–117. DOI: 10.1109/CNSM.2014.7014147.

[38] O. I. Bentstuen and J. Flathagen. "On Bootstrapping In-Band Control Channels in Software Defined Networks." In: *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. May 2018, pp. 1–6. DOI: 10.1109/ICCW.2018.8403796.

[39] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and et al. "ONOS: Towards an Open, Distributed SDN OS." In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 1–6. ISBN: 9781450329897. DOI: 10.1145/2620728.2620744. URL: https://doi.org/10.1145/2620728.2620744.

[40] M. Bertier, O. Marin, and P. Sens. "Implementation and performance evaluation of an adaptable failure detector." In: *Proceedings International Conference on Dependable Systems and Networks*. June 2002, pp. 354–363. DOI: 10.1109/DSN.2002.1028920.

[41] A. Bessani, J. Sousa, and E. E. P. Alchieri. "State Machine Replication for the Masses with BFT-SMART." In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. June 2014, pp. 355–362. DOI: 10.1109/DSN.2014.43.

[42] R. Bifulco and A. Matsiuk. "Towards Scalable SDN Switches: Enabling Faster Flow Table Entries Installation." In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: Association for Computing Machinery, 2015, pp. 343–344. ISBN: 9781450335423. DOI: 10.1145/2785956.2790008. URL: https://doi.org/10.1145/2785956.2790008.

[43] A. Blenk and W. Kellerer. "Towards Virtualization of Software-Defined Networks: A Journey in Three Acts." In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Apr. 2019, pp. 677–682.

[44] G. Bolch, S. Greiner, H. Meer, and K. Trivedi. *Queuing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Jan. 2006. ISBN: 978-0471565253. DOI: 10.1002/0471200581.ch6.

[45] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, and P. Li. “Paxos Replicated State Machines as the Basis of a High-Performance Data Store.” In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI’11. Boston, MA: USENIX Association, 2011, pp. 141–154.

[46] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and et al. “P4: Programming Protocol-Independent Packet Processors.” In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[47] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. V. Ramos, and A. Bessani. “Design and Implementation of a Consistent Data Store for a Distributed SDN Control Plane.” In: *2016 12th European Dependable Computing Conference (EDCC)*. Sept. 2016, pp. 169–180. DOI: 10.1109/EDCC.2016.12.

[48] M. Burrows. “The Chubby Lock Service for Loosely-Coupled Distributed Systems.” In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI ’06. Seattle, Washington: USENIX Association, 2006, pp. 335–350. ISBN: 1931971471.

[49] C. Cachin et al. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011. ISBN: 978-3-642-15259-7. DOI: 10.1007/978-3-642-15260-3_5.

[50] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid. “A Self-Organizing Distributed and In-Band SDN Control Plane.” In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. June 2017, pp. 2656–2657. DOI: 10.1109/ICDCS.2017.328.

[51] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid. “Renaissance: A Self-Stabilizing Distributed SDN Control Plane.” In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 233–243. DOI: 10.1109/ICDCS.2018.00032.

[52] D. Caro. *Automation Network Selection: A Reference Manual, 2nd Edition*. 2nd. Research Triangle Park, NC, USA: International Society of Automation, 2009. ISBN: 1934394890. DOI: 10.5555/1538578.

[53] M. Castro and B. Liskov. “Practical Byzantine Fault Tolerance.” In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI ’99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1880446391. DOI: 10.5555/296806.296824.

[54] M. Castro and B. Liskov. “Practical Byzantine Fault Tolerance and Proactive Recovery.” In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 398–461. ISSN: 0734-2071. DOI: 10.1145/571637.571640. URL: https://doi.org/10.1145/571637.571640.

[55] T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, and W. H. Sanders. “The Mobius modeling environment: recent developments.” In: *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings*. Sept. 2004, pp. 328–329. DOI: 10.1109/QEST.2004.1348051.

[56] T. Cruz, P. Simões, and E. Monteiro. "Virtualizing Programmable Logic Controllers: Toward a Convergent Approach." In: *IEEE Embedded Systems Letters* 8.4 (Dec. 2016), pp. 69–72. ISSN: 1943-0671. DOI: 10.1109/LES.2016.2608418.

[57] H. T. Dang, M. Canini, F. Pedone, and R. Soulé. "Paxos Made Switch-y." In: *SIGCOMM Comput. Commun. Rev.* 46.2 (May 2016), pp. 18–24. ISSN: 0146-4833. DOI: 10.1145/2935634.2935638. URL: https://doi.org/10.1145/2935634.2935638.

[58] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. "Dynamo: Amazon's Highly Available Key-Value Store." In: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: Association for Computing Machinery, 2007, pp. 205–220. ISBN: 9781595935915. DOI: 10.1145/1294261.1294281. URL: https://doi.org/10.1145/1294261.1294281.

[59] T. S. Denis and S. Johnson. "Chapter 6 - Message - Authentication Code Algorithms." In: *Cryptography for Developers*. Ed. by T. S. Denis and S. Johnson. Burlington: Syngress, pp. 251–296. ISBN: 978-1-59749-104-4. DOI: 10.5555/1209478.

[60] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs." In: *Numer. Math.* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[61] T. Distler, C. Cachin, and R. Kapitza. "Resource-Efficient Byzantine Fault Tolerance." In: *IEEE Transactions on Computers* 65.9 (Sept. 2016), pp. 2807–2819. ISSN: 2326-3814. DOI: 10.1109/TC.2015.2495213.

[62] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg. "Byzantine Fault Tolerance, from Theory to Reality." In: *Computer Safety, Reliability, and Security*. Ed. by S. Anderson, M. Felici, and B. Littlewood. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 235–248. ISBN: 978-3-540-39878-3. DOI: 10.1007/978-3-540-39878-3_19.

[63] L. Duerkop, H. Trsek, J. Jasperneite, and L. Wisniewski. "Towards autoconfiguration of industrial automation systems: A case study using Profinet IO." In: *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*. Sept. 2012, pp. 1–8. DOI: 10.1109/ETFA.2012.6489654.

[64] M. Ehrlich, D. Krummacker, C. Fischer, R. Guillaume, S. S. Perez Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, H. D. Schotten, and J. Jasperneite. "Software- Defined Networking as an Enabler for Future Industrial Network Management." In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. Sept. 2018, pp. 1109–1112. DOI: 10.1109/ETFA.2018.8502561.

[65] M. Eischer and T. Distler. "Scalable Byzantine Fault Tolerance on Heterogeneous Servers." In: *2017 13th European Dependable Computing Conference (EDCC)*. Sept. 2017, pp. 34–41. DOI: 10.1109/EDCC.2017.15.

[66] K. Elmeleegy, A. L. Cox, and T. S. E. Ng. “Understanding and Mitigating the Effects of Count to Infinity in Ethernet Networks.” In: *IEEE/ACM Transactions on Networking* 17.1 (Feb. 2009), pp. 186–199. ISSN: 1558-2566. DOI: 10.1109/TNET.2008.920874.

[67] N. Finn. “Introduction to Time-Sensitive Networking.” In: *IEEE Communications Standards Magazine* 2.2 (June 2018), pp. 22–28. ISSN: 2471-2833. DOI: 10.1109/MCOMSTD.2018.1700076.

[68] S. Gilbert and N. Lynch. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services.” In: *SIGACT News* 33.2 (June 2002), pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601. URL: https://doi.org/10.1145/564585.564601.

[69] M. Giles. “Chapter 10 - Approximating the erfinv Function.” In: *GPU Computing Gems Jade Edition*. Ed. by W.-m. W. Hwu. Applications of GPU Computing Series. Boston: Morgan Kaufmann, 2012, pp. 109–116. ISBN: 978-0-12-385963-1. DOI: https://doi.org/10.1016/B978-0-12-385963-1.00010-1. URL: http://www.sciencedirect.com/science/article/pii/B9780123859631000101.

[70] P. Goltsmann, M. Zitterbart, A. Hecker, and R. Bless. *Towards a resilient in-band SDN control channel*. Universitätsbibliothek Tübingen, 2017. DOI: 10.15496/publikation-19547.

[71] A. Gonzalez, P. Gronsund, K. Mahmood, B. Helvik, P. Heegaard, and G. Nencioni. “Service Availability in the NFV Virtualized Evolved Packet Core.” In: *2015 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2015.7417254.

[72] B. Görkemli, S. Tatlıcıoğlu, A. M. Tekalp, S. Civanlar, and E. Lokman. “Dynamic Control Plane for SDN at Scale.” In: *IEEE Journal on Selected Areas in Communications* 36.12 (Dec. 2018), pp. 2688–2701. ISSN: 1558-0008. DOI: 10.1109/JSAC.2018.2871308.

[73] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. “Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure.” In: *Proceedings of the 2016 ACM SIGCOMM Conference*. SIGCOMM ’16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 58–72. ISBN: 9781450341936. DOI: 10.1145/2934872.2934891. URL: https://doi.org/10.1145/2934872.2934891.

[76] J. W. Guck, M. Reisslein, and W. Kellerer. “Function Split Between Delay-Constrained Routing and Resource Allocation for Centrally Managed QoS in Industrial Networks.” In: *IEEE Transactions on Industrial Informatics* 12.6 (2016), pp. 2050–2061. DOI: 10.1109/TII.2016.2592481.

[74] J. W. Guck, A. Van Bemten, and W. Kellerer. “DetServ: Network Models for Real-Time QoS Provisioning in SDN-Based Industrial Environments.” In: *IEEE Transactions on Network and Service Management* 14.4 (Dec. 2017), pp. 1003–1017. ISSN: 2373-7379. DOI: 10.1109/TNSM.2017.2755769.

[75] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer. “Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation.” In: *IEEE Communications Surveys Tutorials* 20.1 (2018), pp. 388–415. ISSN: 2373-745X. DOI: 10.1109/COMST.2017.2749760.

[77] S. Hamadi, I. Snaiki, and O. Cherkaoui. “Fast path acceleration for open vSwitch in overlay networks.” In: *2014 Global Information Infrastructure and Networking Symposium (GIIS)*. Sept. 2014, pp. 1–5.

[78] R. Hanmer, L. Jagadeesan, V. Mendiratta, and H. Zhang. “Friend or Foe: Strong Consistency vs. Overload in High-Availability Distributed Systems and SDN.” In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Oct. 2018, pp. 59–64. DOI: 10.1109/ISSREW.2018.00-30.

[79] N. Hayashibara, A. Cherif, and T. Katayama. “Failure detectors for large-scale distributed systems.” In: *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings*. Oct. 2002, pp. 404–409. DOI: 10.1109/RELDIS.2002.1180218.

[80] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. “The /spl phi/ accrual failure detector.” In: *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004*. Oct. 2004, pp. 66–78. DOI: 10.1109/RELDIS.2004.1353004.

[81] P. E. Heegaard and K. S. Trivedi. “Survivability modeling with stochastic reward nets.” In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. Dec. 2009, pp. 807–818. DOI: 10.1109/WSC.2009.5429679.

[82] P. Heise, F. Geyer, and R. Obermaisser. “Self-configuring deterministic network with in-band configuration channel.” In: *2017 Fourth International Conference on Software Defined Systems (SDS)*. May 2017, pp. 162–167. DOI: 10.1109/SDS.2017.7939158.

[83] P. Heise, M. Lasch, F. Geyer, and R. Obermaisser. “Self-configuring real-time communication network based on OpenFlow.” In: *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. June 2016, pp. 1–6. DOI: 10.1109/LANMAN.2016.7548851.

[84] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr. “Scaling TSN Scheduling for Factory Automation Networks.” In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020, pp. 1–8. DOI: 10.1109/WFCS47810.2020.9114415.

[85] D. Henneke, L. Wisniewski, and J. Jasperneite. “Analysis of realizing a future industrial network by means of Software-Defined Networking (SDN).” In: *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. May 2016, pp. 1–4. DOI: 10.1109/WFCS.2016.7496525.

[86] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia. “POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks.” In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. May 2014, pp. 1–2. DOI: 10.1109/NOMS.2014.6838275.

[87] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft. “Raft Refloated: Do We Have Consensus?” In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 12–21. ISSN: 0163-5980. DOI: 10.1145/2723872.2723876. URL: https://doi.org/10.1145/2723872.2723876.

[88] J. Hu, C. Lin, X. Li, and J. Huang. "Scalability of control planes for Software defined networks: Modeling and evaluation." In: *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*. May 2014, pp. 147–152. DOI: 10.1109/IWQoS.2014.6914314.

[89] X. Huang, S. Bian, Z. Shao, and H. Xu. "Dynamic switch-controller association and control devolution for SDN systems." In: *2017 IEEE International Conference on Communications (ICC)*. May 2017, pp. 1–6. DOI: 10.1109/ICC.2017.7997427.

[90] Y. Huang, J. Tang, Y. Cheng, H. Li, K. A. Campbell, and Z. Han. "Real-Time Detection of False Data Injection in Smart Grid Networks: An Adaptive CUSUM Method and Analysis." In: *IEEE Systems Journal* 10.2 (June 2016), pp. 532–543. ISSN: 2373-7816. DOI: 10.1109/JSYST.2014.2323266.

[92] "IEEE Standard for Local and metropolitan area networks— Bridges and Bridged Networks - Amendment 24: Path Control and Reservation." In: *IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q-2014/Cor 1-2015)* (Mar. 2016), pp. 1–120. ISSN: null. DOI: 10.1109/IEEESTD.2016.7434544.

[93] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements." In: *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)* (Oct. 2018), pp. 1–208. ISSN: null. DOI: 10.1109/IEEESTD.2018.8514112.

[94] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville. "Refining Network Intents for Self-Driving Networks." In: *Proceedings of the Afternoon Workshop on Self-Driving Networks*. SelfDN 2018. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 15–21. ISBN: 9781450359146. DOI: 10.1145/3229584.3229590. URL: https://doi.org/10.1145/3229584.3229590.

[95] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer. "Interfaces, attributes, and use cases: A compass for SDN." In: *IEEE Communications Magazine* 52.6 (June 2014), pp. 210–217. ISSN: 1558-1896. DOI: 10.1109/MCOM.2014.6829966.

[96] D.-C. Juan, L. Li, H.-K. Peng, D. Marculescu, and C. Faloutsos. "Beyond Poisson: Modeling Inter-Arrival Time of Requests in a Datacenter." In: *Advances in Knowledge Discovery and Data Mining*. Ed. by V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, and H.-Y. Kao. Cham: Springer International Publishing, 2014, pp. 198–209. ISBN: 978-3-319-06605-9. DOI: 10.1007/978-3-319-06605-9_17.

[97] H.-J. Jung and S.-J. Hong. "The Quality Control of Software Reliability Based on Functionality, Reliability and Usability." In: *Future Generation Information Technology*. Ed. by T.-h. Kim, Y.-h. Lee, and W.-c. Fang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 112–118. ISBN: 978-3-642-35585-1. DOI: 10.1007/978-3-642-35585-1_15.

[98] M. Karakus and A. Durresi. “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN).” In: *Computer Networks* 112 (2017), pp. 279–293. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2016.11.017. URL: http://www.sciencedirect.com/science/article/pii/S138912861630411X.

[99] R. Katiyar, P. Pawar, A. Gupta, and K. Kataoka. “Auto-Configuration of SDN Switches in SDN/Non-SDN Hybrid Network.” In: *Proceedings of the Asian Internet Engineering Conference*. AINTEC ’15. Bangkok, Thailand: Association for Computing Machinery, 2015, pp. 48–53. ISBN: 9781450339148. DOI: 10.1145/2837030.2837037. URL: https://doi.org/10.1145/2837030.2837037.

[100] N. Katta, H. Zhang, M. Freedman, and J. Rexford. “Ravana: Controller Fault-Tolerance in Software-Defined Networking.” In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. SOSR ’15. Santa Clara, California: Association for Computing Machinery, 2015. ISBN: 9781450334518. DOI: 10.1145/2774993.2774996. URL: https://doi.org/10.1145/2774993.2774996.

[101] A. Katti, G. Di Fatta, T. Naughton, and C. Engelmann. “Scalable and Fault Tolerant Failure Detection and Consensus.” In: *Proceedings of the 22nd European MPI Users’ Group Meeting*. EuroMPI ’15. Bordeaux, France: Association for Computing Machinery, 2015. ISBN: 9781450337953. DOI: 10.1145/2802658.2802660. URL: https://doi.org/10.1145/2802658.2802660.

[102] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. “VeriFlow: Verifying Network-Wide Invariants in Real Time.” In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN ’12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 49–54. ISBN: 9781450314770. DOI: 10.1145/2342441.2342452. URL: https://doi.org/10.1145/2342441.2342452.

[103] P. Koopman. “Guest Editor’s Introduction: Critical Embedded Automotive Networks.” In: *IEEE Micro* 22.04 (July 2002), pp. 14–18. ISSN: 1937-4143. DOI: 10.1109/MM.2002.1028471.

[104] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and et al. “Onix: A Distributed Control Platform for Large-Scale Production Networks.” In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI’10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 351–364. DOI: 10.5555/1924943.1924968.

[105] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. “Zyzzyva: Speculative Byzantine Fault Tolerance.” In: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. SOSP ’07. Stevenson, Washington, USA: Association for Computing Machinery, 2007, pp. 45–58. ISBN: 9781595935915. DOI: 10.1145/1294261.1294267. URL: https://doi.org/10.1145/1294261.1294267.

[106] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2371999.

[107] S. Krishnamurthy, W. H. Sanders, and M. Cukier. “An adaptive framework for tunable consistency and timeliness using replication.” In: *Proceedings International Conference on Dependable Systems and Networks*. June 2002, pp. 17–26. DOI: 10.1109/DSN.2002.1028882.

[108] M. Kuźniar, P. Perešíni, and D. Kostić. “What You Need to Know About SDN Flow Tables.” In: *Passive and Active Measurement*. Ed. by J. Mirkovic and Y. Liu. Cham: Springer International Publishing, 2015, pp. 347–359. ISBN: 978-3-319-15509-8. DOI: 10.1007/978-3-319-15509-8_26.

[109] L. Lamport and M. Massa. “Cheap Paxos.” In: *International Conference on Dependable Systems and Networks, 2004*. June 2004, pp. 307–314. DOI: 10.1109/DSN.2004.1311900.

[110] L. Lamport. “Fast Paxos.” In: *Distributed Computing* 19.2 (Oct. 2006), pp. 79–103. ISSN: 1432-0452. DOI: 10.1007/s00446-006-0005-x. URL: https://doi.org/10.1007/s00446-006-0005-x.

[111] L. Lamport. “The Part-Time Parliament.” In: *ACM Trans. Comput. Syst.* 16.2 (May 1998), pp. 133–169. ISSN: 0734-2071. DOI: 10.1145/279227.279229. URL: https://doi.org/10.1145/279227.279229.

[112] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. “Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks.” In: *IEEE Transactions on Network and Service Management* 12.1 (Mar. 2015), pp. 4–17. ISSN: 2373-7379. DOI: 10.1109/TNSM.2015.2402432.

[113] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. “Logically Centralized? State Distribution Trade-Offs in Software Defined Networks.” In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN ’12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 1–6. ISBN: 9781450314770. DOI: 10.1145/2342441.2342443. URL: https://doi.org/10.1145/2342441.2342443.

[114] D. Li, M. Zhou, P. Zeng, M. Yang, Y. Zhang, and H. Yu. “Green and reliable software-defined industrial networks.” In: *IEEE Communications Magazine* 54.10 (Oct. 2016), pp. 30–37. ISSN: 1558-1896. DOI: 10.1109/MCOM.2016.7588226.

[115] H. Li, P. Li, S. Guo, and A. Nayak. “Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud.” In: *IEEE Transactions on Cloud Computing* 2.4 (Oct. 2014), pp. 436–447. ISSN: 2372-0018. DOI: 10.1109/TCC.2014.2355227.

[116] J. Liu, W. Li, G. O. Karame, and N. Asokan. “Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing.” In: *IEEE Transactions on Computers* 68.1 (Jan. 2019), pp. 139–151. ISSN: 2326-3814. DOI: 10.1109/TC.2018.2860009.

[117] Y. Liu, Y. Ma, J. J. Han, H. Levendel, and K. S. Trivedi. “A proactive approach towards always-on availability in broadband cable networks.” In: *Computer Communications* 28.1 (2005), pp. 51–64. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2004.08.022. URL: http://www.sciencedirect.com/science/article/pii/S0140366404003202.

[118] F. Machida, V. F. Nicola, and K. S. Trivedi. “Job Completion Time on a Virtualized Server with Software Rejuvenation.” In: *J. Emerg. Technol. Comput. Syst.* 10.1 (Jan. 2014). ISSN: 1550-4832. DOI: 10.1145/2539121. URL: https://doi.org/10.1145/2539121.

[119] T. Mahmoodi, V. Kulkarni, W. Kellerer, P. Mangan, F. Zeiger, S. Spirou, I. Askoxylakis, X. Vilajosana, H. J. Einsiedler, and J. Quittek. “VirtuWind: virtual and programmable industrial network prototype deployed in operational wind park.” In: *Transactions on Emerging Telecommunications Technologies* 27.9 (2016), pp. 1281–1288. DOI: 10.1002/ett.3057. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.3057. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3057.

[120] M. Malhotra, J. K. Muppala, and K. S. Trivedi. “Stiffness-tolerant methods for transient analysis of stiff Markov chains.” In: *Microelectronics Reliability* 34.11 (1994), pp. 1825–1841. ISSN: 0026-2714. DOI: https://doi.org/10.1016/0026-2714(94)90137-6. URL: http://www.sciencedirect.com/science/article/pii/0026271494901376.

[121] Y. Mao, F. P. Junqueira, and K. Marzullo. “Mencius: Building Efficient Replicated State Machines for WANs.” In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*. OSDI’08. San Diego, California: USENIX Association, 2008, pp. 369–384.

[122] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. “OpenFlow: Enabling Innovation in Campus Networks.” In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[123] J. Medved, R. Varga, A. Tkacik, and K. Gray. “OpenDaylight: Towards a Model-Driven SDN Controller architecture.” In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. June 2014, pp. 1–6. DOI: 10.1109/WoWMoM.2014.6918985.

[124] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman. “Application-Aware Data Plane Processing in SDN.” In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN ’14. Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 13–18. ISBN: 9781450329897. DOI: 10.1145/2620728.2620735. URL: https://doi.org/10.1145/2620728.2620735.

[125] V. B. Mendiratta, L. J. Jagadeesan, R. Hanmer, and M. R. Rahman. “How Reliable Is My Software-Defined Network? Models and Failure Impacts.” In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Oct. 2018, pp. 83–88. DOI: 10.1109/ISSREW.2018.00-26.

[126] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. “The Honey Badger of BFT Protocols.” In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 31–42. ISBN: 9781450341394. DOI: 10.1145/2976749.2978399. URL: https://doi.org/10.1145/2976749.2978399.

[127] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese. "Application Performance Pitfalls and TCP's Nagle Algorithm." In: *SIGMETRICS Perform. Eval. Rev.* 27.4 (Mar. 2000), pp. 36–44. ISSN: 0163-5999. DOI: 10.1145/346000.346012. URL: https://doi.org/10.1145/346000.346012.

[128] P. M. Mohan, T. Truong-Huu, and M. Gurusamy. "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks." In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–7. DOI: 10.1109/GLOCOM.2017.8254755.

[129] P. M. Mohan, T. Truong-Huu, and M. Gurusamy. "Towards resilient in-band control path routing with malicious switch detection in SDN." In: *2018 10th International Conference on Communication Systems Networks (COMSNETS)*. Jan. 2018, pp. 9–16. DOI: 10.1109/COMSNETS.2018.8328174.

[130] E. Molina, E. Jacob, N. Toledo, and A. Astarloa. "Performance Enhancement of High-Availability Seamless Redundancy (HSR) Networks Using OpenFlow." In: *IEEE Communications Letters* 20.2 (Feb. 2016), pp. 364–367. ISSN: 2373-7891. DOI: 10.1109/LCOMM.2015.2504442.

[131] E. Molina et al. "Availability improvement of Layer 2 seamless networks using Openflow." In: *The Scientific World Journal* 2015 (2015). DOI: 10.1155/2015/283165.

[132] I. Moraru, D. G. Andersen, and M. Kaminsky. "There is More Consensus in Egalitarian Parliaments." In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: Association for Computing Machinery, 2013, pp. 358–372. ISBN: 9781450323888. DOI: 10.1145/2517349.2517350. URL: https://doi.org/10.1145/2517349.2517350.

[133] I. Moraru, D. G. Andersen, and M. Kaminsky. "There is More Consensus in Egalitarian Parliaments." In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: Association for Computing Machinery, 2013, pp. 358–372. ISBN: 9781450323888. DOI: 10.1145/2517349.2517350. URL: https://doi.org/10.1145/2517349.2517350.

[134] G. Morel, P. Valckenaers, J.-M. Faure, C. E. Pereira, and C. Diedrich. "Manufacturing plant control challenges and issues." In: *Control Engineering Practice* 15.11 (2007). Special Issue on Manufacturing Plant Control: Challenges and Issues, pp. 1321–1331. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2007.05.005.

[135] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier. "Inter-controller traffic in ONOS clusters for SDN networks." In: *2016 IEEE International Conference on Communications (ICC)*. May 2016, pp. 1–6. DOI: 10.1109/ICC.2016.7511034.

[136] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier. "Inter-Controller Traffic to Support Consistency in ONOS Clusters." In: *IEEE Transactions on Network and Service Management* 14.4 (Dec. 2017), pp. 1018–1031. ISSN: 2373-7379. DOI: 10.1109/TNSM.2017.2723477.

[137] K. Nagaraja, N. Krishnan, R. Bianchini, R. P. Martin, and T. D. Nguyen. "Quantifying and Improving the Availability of High-Performance Cluster-Based Internet Services." In: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. SC '03. Phoenix, AZ, USA: Association for Computing Machinery, 2003, p. 27. ISBN: 1581136951. DOI: 10.1145/1048935.1050178. URL: https://doi.org/10.1145/1048935.1050178.

[138] K. Naik and B. Vidhya. "A Group Tasks Scheduling Algorithm for Cloud Computing Networks based on QoS." In: *International Journal of Engineering and Technology(UAE)* 7 (Sept. 2018), pp. 53–59. DOI: 10.14419/ijet.v7i4.6.20236.

[139] N. Naik. "Applying Computational Intelligence for enhancing the dependability of multi-cloud systems using Docker Swarm." In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. Dec. 2016, pp. 1–7. DOI: 10.1109/SSCI.2016.7850194.

[140] M. A. A. Ndiaye, J. Petin, J. Camerini, and J. P. Georges. "Performance assessment of industrial control system during pre-sales uncertain context using automatic Colored Petri Nets model generation." In: *2016 International Conference on Control, Decision and Information Technologies (CoDIT)*. Apr. 2016, pp. 671–676. DOI: 10.1109/CoDIT.2016.7593643.

[141] G. Nencioni, B. E. Helvik, A. J. Gonzalez, P. E. Heegaard, and A. Kamisinski. "Availability Modelling of Software-Defined Backbone Networks." In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. June 2016, pp. 105–112. DOI: 10.1109/DSN-W.2016.28.

[142] G. Nencioni, B. E. Helvik, and P. E. Heegaard. "Including Failure Correlation in Availability Modeling of a Software-Defined Backbone Network." In: *IEEE Transactions on Network and Service Management* 14.4 (2017), pp. 1032–1045. DOI: 10.1109/TNSM.2017.2755462.

[143] H. V. Netto, L. C. Lung, M. Correia, A. F. Luiz, and L. M. S. de Souza. "State machine replication in containers managed by Kubernetes." In: *Journal of Systems Architecture* 73 (2017). Special Issue on Reliable Software Technologies for Dependable Distributed Systems, pp. 53–59. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2016.12.007. URL: http://www.sciencedirect.com/science/article/pii/S1383762116302752.

[144] J. Núñez-Martínez, J. Baranda, and J. Mangues-Bafalluy. "A service-based model for the hybrid software defined wireless mesh backhaul of small cells." In: *2015 11th International Conference on Network and Service Management (CNSM)*. Nov. 2015, pp. 390–393. DOI: 10.1109/CNSM.2015.7367388.

[145] B. M. Oki and B. H. Liskov. "Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems." In: *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*. PODC '88. Toronto, Ontario, Canada: Association for Computing Machinery, 1988, pp. 8–17. ISBN: 0897912772. DOI: 10.1145/62546.62549. URL: https://doi.org/10.1145/62546.62549.

[146] D. Ongaro and J. Ousterhout. “In Search of an Understandable Consensus Algorithm.” In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 305–319. ISBN: 978-1-931971-10-2. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[147] M. Pahlevan and R. Obermaisser. “Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks.” In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2018, pp. 337–344. DOI: 10.1109/ETFA.2018.8502515.

[148] N. Paladi et al. “TruSDN: Bootstrapping Trust in Cloud Network Infrastructure.” In: *International Conference on Security and Privacy in Communication Systems*. Springer. Springer, Cham, 2016. ISBN: 978-3-319-59607-5. DOI: 10.1007/978-3-319-59608-2_6.

[149] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker. “CAP for Networks.” In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN ’13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 91–96. ISBN: 9781450321785. DOI: 10.1145/2491185.2491186. URL: https://doi.org/10.1145/2491185.2491186.

[150] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker. “SCL: Simplifying Distributed SDN Control Planes.” In: *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*. NSDI’17. Boston, MA, USA: USENIX Association, 2017, pp. 329–345. ISBN: 9781931971379.

[151] P. Patil, A. Gokhale, and A. Hakiri. “Bootstrapping Software Defined Network for flexible and dynamic control plane management.” In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. Apr. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7116132.

[152] N. E. Petroulakis, K. Fysarakis, I. Askoxylakis, and G. Spanoudakis. “Reactive security for SDN/NFV-enabled industrial networks leveraging service function chaining.” In: *Transactions on Emerging Telecommunications Technologies* 29.7 (2018). e3269 ett.3269, e3269. DOI: 10.1002/ett.3269.

[153] T. Pfeiffenberger and J. L. Du. “Evaluation of software-defined networking for power systems.” In: *2014 IEEE International Conference on Intelligent Energy and Power Systems (IEPS)*. June 2014, pp. 181–185. DOI: 10.1109/IEPS.2014.6874175.

[154] S. Qian, F. Luo, and J. Xu. “An Analysis of Frame Replication and Elimination for Time-Sensitive Networking.” In: *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*. ICNCC 2017. Kunming, China: Association for Computing Machinery, 2017, pp. 166–170. ISBN: 9781450353663. DOI: 10.1145/3171592.3171614. URL: https://doi.org/10.1145/3171592.3171614.

[155] S. Ranganathan, A. D. George, R. W. Todd, and M. C. Chidester. “Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters.” In: *Cluster Computing* 4.3 (July 2001), pp. 197–209. ISSN: 1386-7857. DOI: 10.1023/A:1011494323443. URL: https://doi.org/10.1023/A:1011494323443.

[156] A. Reibman and K. Trivedi. "Numerical transient analysis of markov models." In: *Computers & Operations Research* 15.1 (1988), pp. 19–36. ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(88)90026-3. URL: http://www.sciencedirect.com/science/article/pii/0305054888900263.

[157] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas. "Efficient Reconciliation and Flow Control for Anti-Entropy Protocols." In: *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. LADIS '08. Yorktown Heights, New York, USA: Association for Computing Machinery, 2008. ISBN: 9781605582962. DOI: 10.1145/1529974.1529983. URL: https://doi.org/10.1145/1529974.1529983.

[158] R. van Renesse, Y. Minsky, and M. Hayden. "A Gossip-Style Failure Detection Service." In: *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. Middleware '98. The Lake District, United Kingdom: Springer-Verlag, 2009, pp. 55–70. ISBN: 1852330880. DOI: 10.5555/1659232.1659238.

[159] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller. "Too Many SDN Rules? Compress Them with MINNIE." In: *2015 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2015, pp. 1–7. DOI: 10.1109/GLOCOM.2015.7417661.

[160] Y. Saito and M. Shapiro. "Optimistic Replication." In: *ACM Comput. Surv.* 37.1 (Mar. 2005), pp. 42–81. ISSN: 0360-0300. DOI: 10.1145/1057977.1057980. URL: https://doi.org/10.1145/1057977.1057980.

[161] W. H. Sanders and J. F. Meyer. "Reduced base model construction methods for stochastic activity networks." In: *IEEE Journal on Selected Areas in Communications* 9.1 (Jan. 1991), pp. 25–36. ISSN: 1558-0008. DOI: 10.1109/49.64901.

[162] W. H. Sanders and J. F. Meyer. "Stochastic Activity Networks: Formal Definitions and Concepts." In: *Lectures on Formal Methods and PerformanceAnalysis: First EEF/Euro Summer School on Trends in Computer Science Bergen Dal, The Netherlands, July 3–7, 2000 Revised Lectures*. Ed. by E. Brinksma, H. Hermanns, and J.-P. Katoen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 315–343. ISBN: 978-3-540-44667-5. DOI: 10.1007/3-540-44667-2_9. URL: https://doi.org/10.1007/3-540-44667-2_9.

[163] R. Santos and A. Kassler. "Small Cell Wireless Backhaul Reconfiguration Using Software-Defined Networking." In: *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. Mar. 2017, pp. 1–6. DOI: 10.1109/WCNC.2017.7925943.

[164] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis. "In-Network Computation is a Dumb Idea Whose Time Has Come." In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. HotNets-XVI. Palo Alto, CA, USA: Association for Computing Machinery, 2017, pp. 150–156. ISBN: 9781450355698. DOI: 10.1145/3152434.3152461. URL: https://doi.org/10.1145/3152434.3152461.

[165] F. Sardis, M. Condoluci, T. Mahmoodi, and M. Dohler. “Can QoS be dynamically manipulated using end-device initialization?” In: *2016 IEEE International Conference on Communications Workshops (ICC)*. May 2016, pp. 448–454. DOI: 10.1109/ICCW.2016.7503828.

[166] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. “A New Adaptive Accrual Failure Detector for Dependable Distributed Systems.” In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC ’07. Seoul, Korea: Association for Computing Machinery, 2007, pp. 551–555. ISBN: 1595934804. DOI: 10.1145/1244002.1244129. URL: https://doi.org/10.1145/1244002.1244129.

[167] L. Schiff, S. Schmid, and M. Canini. “Ground Control to Major Faults: Towards a Fault Tolerant and Adaptive SDN Control Network.” In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. June 2016, pp. 90–96. DOI: 10.1109/DSN-W.2016.48.

[168] L. Schiff, S. Schmid, and M. Canini. “Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network.” In: *ACM SIGCOMM Symposium on SDN Research (SOSR)*. New York, NY, USA: Association for Computing Machinery, 2015.

[169] L. Schiff, S. Schmid, and P. Kuznetsov. “In-Band Synchronization for Distributed SDN Control Planes.” In: *SIGCOMM Comput. Commun. Rev.* 46.1 (Jan. 2016), pp. 37–43. ISSN: 0146-4833. DOI: 10.1145/2875951.2875957. URL: https://doi.org/10.1145/2875951.2875957.

[170] F. B. Schneider. “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial.” In: *ACM Comput. Surv.* 22.4 (Dec. 1990), pp. 299–319. ISSN: 0360-0300. DOI: 10.1145/98163.98167. URL: https://doi.org/10.1145/98163.98167.

[171] S. Scott-Hayward. “Design and deployment of secure, robust, and resilient SDN controllers.” In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. Apr. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7258233.

[172] R. Serna Oliver, S. S. Craciunas, and W. Steiner. “IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding.” In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2018, pp. 13–24. DOI: 10.1109/RTAS.2018.00008.

[173] M. A. Shahin. “Smart Grid self-healing implementation for underground distribution networks.” In: *2013 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*. Nov. 2013, pp. 1–5. DOI: 10.1109/ISGT-Asia.2013.6698702.

[174] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. “Conflict-Free Replicated Data Types.” In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. SSS’11. Grenoble, France: Springer-Verlag, 2011, pp. 386–400. ISBN: 9783642245497. DOI: 10.5555/2050613.2050642.

[175] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. “A demonstration of automatic bootstrapping of resilient OpenFlow networks.” In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. May 2013, pp. 1066–1067.

[176] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. “Automatic bootstrapping of OpenFlow networks.” In: *2013 19th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*. Apr. 2013, pp. 1–6. DOI: 10.1109/LANMAN.2013.6528283.

[177] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. “Fast failure recovery for in-band OpenFlow networks.” In: *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*. Mar. 2013, pp. 52–59.

[178] D. Skeen. “Nonblocking Commit Protocols.” In: *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’81. Ann Arbor, Michigan: Association for Computing Machinery, 1981, pp. 133–142. ISBN: 0897910400. DOI: 10.1145/582318.582339. URL: https://doi.org/10.1145/582318.582339.

[179] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll. “RACE: A Centralized Platform Computer Based Architecture for Automotive Applications.” In: *2013 IEEE International Electric Vehicle Conference (IEVC)*. Oct. 2013, pp. 1–6. DOI: 10.1109/IEVC.2013.6681152.

[180] Y. Su, I. Wang, Y. Hsu, and C. H. .-. Wen. “FASIC: A Fast-Recovery, Adaptively Spanning In-Band Control Plane in Software-Defined Network.” In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254760.

[181] D. Suh, S. Jang, S. Han, S. Pack, M. Kim, T. Kim, and C. Lim. “Toward Highly Available and Scalable Software Defined Networks for Service Providers.” In: *IEEE Communications Magazine* 55.4 (Apr. 2017), pp. 100–107. ISSN: 1558-1896. DOI: 10.1109/MCOM.2017.1600170.

[182] D. Suh, S. Jang, S. Han, S. Pack, T. Kim, and J. Kwak. “On performance of OpenDaylight clustering.” In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. June 2016, pp. 407–410. DOI: 10.1109/NETSOFT.2016.7502476.

[183] A. Tootoonchian and Y. Ganjali. “HyperFlow: A Distributed Control Plane for OpenFlow.” In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. INM/WREN’10. San Jose, CA: USENIX Association, 2010, p. 3.

[184] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou. “On the placement of management and control functionality in software defined networks.” In: *2015 11th International Conference on Network and Service Management (CNSM)*. Nov. 2015, pp. 360–365. DOI: 10.1109/CNSM.2015.7367383.

[185] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. “Analysis and Implementation of Software Rejuvenation in Cluster Systems.” In: *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’01. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2001, pp. 62–71. ISBN: 1581133340. DOI: 10.1145/378420.378434. URL: https://doi.org/10.1145/378420.378434.

[186] S. Verbrugge, D. Colle, P. Demeester, R. Huelsermann, and M. Jaeger. “General availability model for multilayer transport networks.” In: *DRCN 2005). Proceedings.5th International Workshop on Design of Reliable Communication Networks, 2005.* Oct. 2005, p. 8. DOI: 10.1109/DRCN.2005.1563848.

[187] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo. “Efficient Byzantine Fault-Tolerance.” In: *IEEE Transactions on Computers* 62.1 (Jan. 2013), pp. 16–30. ISSN: 2326-3814. DOI: 10.1109/TC.2011.221.

[188] J. Vestin, A. Kassler, and J. Akerberg. “Resilient software defined networking for industrial control networks.” In: *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*. Dec. 2015, pp. 1–5. DOI: 10.1109/ICICS.2015.7459981.

[189] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer, and C. M. Machuca. “Characterization of failure dynamics in SDN controllers.” In: *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*. Sept. 2017, pp. 1–7. DOI: 10.1109/RNDM.2017.8093037.

[190] D. von Rohr, M. Felser, et al. “Simplifying the engineering of modular PROFINET IO devices.” In: *ETFA2011*. Sept. 2011, pp. 1–4. DOI: 10.1109/ETFA.2011.6059156.

[191] Wei Chen, S. Toueg, and M. Kawazoe Aguilera. “On the quality of service of failure detectors.” In: *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*. June 2000, pp. 191–200. DOI: 10.1109/ICDSN.2000.857535.

[192] M. Wollschlaeger, T. Sauter, and J. Jasperneite. “The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0.” In: *IEEE Industrial Electronics Magazine* 11.1 (Mar. 2017), pp. 17–27. ISSN: 1941-0115. DOI: 10.1109/MIE.2017.2649104.

[193] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv. “Control plane of software defined networks: A survey.” In: *Computer Communications* 67 (2015), pp. 1–10. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2015.06.004.

[194] W. Xie, Y. Hong, and K. S. Trivedi. “Software rejuvenation policies for cluster systems under varying workload.” In: *10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004. Proceedings.* Mar. 2004, pp. 122–129. DOI: 10.1109/PRDC.2004.1276563.

[195] Q. Yan, F. R. Yu, Q. Gong, and J. Li. “Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges.” In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 602–622. ISSN: 2373-745X. DOI: 10.1109/COMST.2015.2487361.

[196] T. Yang and K. Wang. “Failure detection service with low mistake rates for SDN controllers.” In: *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. Oct. 2016, pp. 1–6. DOI: 10.1109/APNOMS.2016.7737210.

[197] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. “Separating Agreement from Execution for Byzantine Fault Tolerant Services.” In: *SIGOPS Oper. Syst. Rev.* 37.5 (Oct. 2003), pp. 253–267. ISSN: 0163-5980. DOI: 10.1145/1165389.945470. URL: https://doi.org/10.1145/1165389.945470.

[198] H. Yu and A. Vahdat. “Design and Evaluation of a Continuous Consistency Model for Replicated Services.” In: *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*. OSDI’00. San Diego, California: USENIX Association, 2000.

[199] T. Zhang, A. Bianco, and P. Giaccone. “The role of inter-controller traffic in SDN controllers placement.” In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Nov. 2016, pp. 87–92. DOI: 10.1109/NFV-SDN.2016.7919481.

[200] Y. Zhang, B. Han, Z.-L. Zhang, and V. Gopalakrishnan. “Network-Assisted Raft Consensus Algorithm.” In: *Proceedings of the SIGCOMM Posters and Demos*. SIGCOMM Posters and Demos ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 94–96. ISBN: 9781450350570. DOI: 10.1145/3123878.3131998. URL: https://doi.org/10.1145/3123878.3131998.

[201] Y. Zhang, E. Ramadan, H. Mekky, and Z.-L. Zhang. “When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network.” In: *Proceedings of the First Asia-Pacific Workshop on Networking*. APNet’17. Hong Kong, China: Association for Computing Machinery, 2017, pp. 1–7. ISBN: 9781450352444. DOI: 10.1145/3106989.3106999. URL: https://doi.org/10.1145/3106989.3106999.

[202] L. Y. Zhi, P. M. Mohan, V. Sridharan, and M. Gurusamy. “Secondary Controller Mapping for Reliable Control Traffic Forwarding in SDN.” In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. July 2018, pp. 1–2. DOI: 10.1109/ICCCN.2018.8487383.

[203] W. Zhou, D. Jin, J. Croft, M. Caesar, and P. B. Godfrey. “Enforcing Customizable Consistency Properties in Software-Defined Networks.” In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 73–85. ISBN: 978-1-931971-218. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/zhou.

Miscellaneous

[204] A. Ademaj, T. Enzinger, and M. Stanica. “TSN System Requirements v0.2.” In: [Online; accessed January-2020]. IEC/IEEE. 2018. URL: http://www.ieee802.org/1/files/public/docs2018/60802-stanica-tsn-system-requirements-0518-v02.pdf.

[205] F. Berkenkamp, A. Krause, and A. P. Schoellig. “Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics.” In: *arXiv preprint arXiv:1602.04450* (2016).

[206] A. Blomdell, I. Dressler, K. Nilsson, and A. Robertsson. *Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers*. eng. [Online; accessed January-2020]. 2010. URL: https://lup.lub.lu.se/search/ws/files/6344117/3626753.pdf.

[207] C. Bowers and J. Farkas. "Applicability of Maximally Redundant Trees to IEEE 802.1 Qca Path Control and Reservation." In: *draft-bowers-rtgwg-mrt-applicability-to-8021qca-00 (work in progress)* (2015).

[208] Canonical. *High-availability SQLite for fault-tolerant IoT and Edge devices*. https://dqlite.io/. [Accessed December-2019].

[209] Cisco Systems, Inc. *Spanning Tree Protocol Problems and Related Design Considerations*. https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/10556-16.html. [Online; accessed January-2020]. 2017.

[210] C. Copeland et al. *Tangaroa: A Byzantine Fault Tolerant Raft*. http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf. [Online; accessed January-2020].

[211] V. Costan and S. Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016 (2016).

[212] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé. *Network hardware-accelerated consensus*. Tech. rep. 2016.

[213] J. Dorr. "IEC/IEEE P60802 JWG TSN Industrial Profile: Use Cases Status Update 2018-05-14." In: [Online; accessed January-2020]. IEC/IEEE. 2018. URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[214] H. Du et al. *Multi-Paxos: An Implementation and Evaluation*. Tech. rep. UW-CSE-09-09-02. Department of Computer Science and Engineering, University of Washington, 2009.

[215] P. Heise. "Real-time guarantees, dependability and self-configuration in future avionic networks." PhD thesis. 2018.

[216] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks." In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), pp. 1–1993.

[217] Internet2 Consortium. *Internet2 Network Infrastructure Topology*. https://www.internet2.edu/media_files/422. [Online; accessed January-2020].

[218] L. Lamport et al. "Paxos made simple." In: *ACM Sigact News* 32.4 (2001).

[219] U. Lehmann. *Redundant control system and control computer and peripheral unit for a control system of this type*. US Patent 7,120,820. Oct. 2006.

[220] National Institute of Standards and Technology. "FIPS 180-2 with change notice, "Secure Hash Standard"." In: (2004).

[221] ON.Lab. *Global SDN Deployment Powered by ONOS*. https://wiki.onosproject.org/display/ONOS/Global+SDN+Deployment+Powered+by+ONOS. [Accessed March-2020].

[222] ON.Lab. *OpenDaylight - Use Cases and Users*. https://www.opendaylight.org/use-cases-and-users. [Accessed March-2020].

[223] D. Ongaro. "Consensus: Bridging theory and practice." PhD thesis. Stanford University, 2014.

[224] P60802 Project: TSN Profile for Industrial Automation (TSN-IA). "Use Cases IEC / IEEE 60802 v1.3." In: [Online; accessed January-2020]. IEC/IEEE. 2018. URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[225] *Pareto Optimal Controller Placement (POCO)*. https://github.com/lsinfo3/poco/tree/master/topologies. [Online; accessed January-2020].

[226] A. Singh et al. *Conflict-free quorum-based BFT protocols*. Tech. rep. 2007.

[227] Sonja Steffens. "Safety @ COTS Multicore: Distributed Smart Safe System DS 3." In: [Online; accessed January-2020]. Siemens Mobility GmbH. 2018. URL: https://smartrail40.ch/service/download.asp?mem=0&path=%5Cdownload%5Cdownloads%5C2018%2011%2013%20Innovationstag%20ETCS%20Stellwerk_smartrail%204.0.pdf.

[228] Specification, OpenFlow Switch. *Version 1.5.1, Standard, Open Networking Foundation*. 2015.

[229] University of Surrey - 5G Innovation Centre. "5G Whitepaper: The Flat Distributed Cloud (FDC) 5G Architecture Revolution." In: (2016).

[230] P. S. Vizarreta. *STSM Modeling of software failures and failure propagation models in Software Defined Networking (SDN)*. Tech. rep. Chair of Communication Networks, Technical University of Munich, 2016.

List of Figures

1.1 Structure of the thesis. 8

3.1 An exemplary industrial SDN with redundant paths for majority of C2C and C2S connections. 20
3.2 A simplified lifecycle schema of a replica inside the Raft cluster. 22
3.3 SAN-based modeling of distributed control plane replica failure. 26
3.4 The SAN-based modeling of distributed control plane replica recovery. 28
3.5 Transient analysis of the SDN controller cluster unavailability over a 1000 hour period. . 30
3.6 Injection of communication link failures leading to false positives in Raft's failure detection. 32
3.7 Link failure injections, local suspicions, replica failure and global agreement. 37

4.1 Raft response time SAN model. 42
4.2 Comparison of experimentally observed and modeled Raft performance with clusters of various sizes. 45
4.3 Probability of an event being successfully handled in a given time period t when deploying Raft. 46
4.4 Probability of receiving an event response during an observation window and occurring replica failures. 46
4.5 Resulting response time assuming an occurrence of $1 \leq N_F \leq (\lfloor \frac{|C|}{2} \rfloor + 1)$ controller failures in a 7-node controller cluster. 47
4.6 Size of the CTMC state space generated using the SAN response time models. 48
4.7 Inaccuracies stemming from a decreased number of Erlang stages E_S used in the approximation of deterministic transitions. 48
4.8 Overhead of the CTMC compilation for varying cluster sizes $|C|$. 49
4.9 Computation time of the *instant of time* [55] transient solutions for evaluated state space sizes. 49
4.10 The proposed event detection reference model. 51
4.11 Impact of selection of the Failure Detection method. 58
4.12 Impact of selection of the Event Agreement method. 59
4.13 Impact of selection of the Dissemination method. 60

5.1 Distributed data stores of SDN controller replicas with varying degrees of state consistency. 64
5.2 High-level architecture of the AC framework. 65

5.3 Exemplary network topologies and controller placements used in the SC, AC and EC evaluation. . . . 74
5.4 The response time of the AC and SC data stores. . . . 76
5.5 PID- and *threshold*-based adaptation of CLs. . . . 77
5.6 ECDFs of reported inefficiencies for various deployed consistency models and request arrival rates in the Fat-Tree topology. . . . 77
5.7 ECDFs of reported consistency-related inefficiencies for the Internet2 topology. . . . 78
5.8 The measured mean inefficiency for different CL parametrizations in AC. . . . 78

6.1 Architecture of MORPH's distributed control plane design. . . . 86
6.2 Visual representation of MORPH's workflow and the distributed algorithm execution. . . . 87
6.3 Impact of injected Byzantine and Fail-Stop failures on C2C, C2S delay, communication overhead and CPU load. . . . 97
6.4 Impact of type of failure on experienced C2C delay in MORPH. . . . 98
6.5 The impact of SDN application's statefulness on C2C and C2S reconfiguration delay. . . . 99
6.6 Switch reconfiguration delay for NON-SELECTIVE and SELECTIVE propagation of controller configurations in MORPH. . . . 99
6.7 Total execution time of ILP solver in MORPH. . . . 100

7.1 Design of MPBFT protocol. . . . 109
7.2 Design of SBFT protocol. . . . 109
7.3 Design of OBFT protocol. . . . 110
7.4 Response time of a BFT-enabled distributed control plane for varied BFT protocols. . . . 114
7.5 Acceptance rates for client requests for varied BFT protocols. . . . 115
7.6 Signaling overhead of various BFT protocols. . . . 115
7.7 P4BFT: Minimization of communication overhead and control plane delay by example. . . . 119
7.8 Packet load improvement of P4BFT over the reference works [1, 115, 128]. . . . 123
7.9 The impact of controller aggregation and placement on the control plane load footprint in a BFT control plane. . . . 123
7.10 The ECDFs of processing delays imposed in a P4BFT's processing node. . . . 124
7.11 ECDFs of switch reconfiguration time in state-of-the-art and P4BFT environments. . . . 124
7.12 Pareto Frontier of P4BFT's solution space. . . . 125
7.13 ILP solution time for random topologies with P4BFT. . . . 125

8.1 HSS - Message sequence diagram of the bootstrapping procedure as described in Section 8.4. 133
8.2 Exemplary resilient path output for $k = 1$ for: (a) S2C paths between `S4` and all controllers; and (b) all C2C pairs. . . . 135
8.3 Exemplary network extension with one switch and two links. c) depicts the resulting tree. 136
8.5 An exemplary data flow of a `FLOW_MOD` RPC with each entity's leader on a different controller. 142
8.6 Considered industrial topologies in evaluation of HSS and HHS [52, 215]. . . . 144
8.7 Observed GBCT for single- and 3-controller scenarios. . . . 144
8.8 TEXT values of the two bootstrapping schemes for configurations deploying 1 to 3 controllers. 145

8.9 Bar charts comparing the average FTO for HSS and HHS bootstrapping schemes. 146

List of Tables

3.1 SAN model parameters used in our solutions. 29

4.1 Parameters used in evaluation of M-EA, L-EA, T-FD, ϕ-FD, RRG-D, BEB-D, UH-S and PR-S. 56

5.1 Notation used in Alg. 1, Alg. 2, Alg. 3, Alg. 4 and Alg. 5. 67
5.2 Parametrizations used in experimental study of different consistency models. 75
5.3 Per-replica load when serving 1000 SDN-LB requests in a 5-controller cluster synchronized using SC, AC or EC. 79

6.1 Notation used in specification of MORPH's communication overhead. 94
6.2 Communication complexity of MORPH. 95
6.3 Parameters used in evaluation of MORPH implementation. 96

7.1 Overview of presented BFT protocols. 106
7.2 Notation used in Tables 7.3, 7.4 and 7.5. 111
7.3 Number of matching messages required to reach consensus for varied BFT protocols. . . 112
7.4 Computational and communication overhead of the introduced BFT protocols. 112
7.5 Constraints used in building the A&E groups. 112
7.6 P4BFT Model Parameters. 118
7.7 Hash table layout of a P4BFT's processing node. 121

8.1 Comparison of our HSS and HHS bootstrapping designs with existing schemes. 130
8.2 HSS - Initial and NEXT flow rules installed on switches throughout `Phase 2`. 134
8.3 HHS - Preconfigured OF rules. 138
8.4 HHS - Initial and NEXT flow rules installed on switches in `Phase 2a`. 139
8.5 Parametrization of the (R)STP timer in HSS. 141
8.6 Controllers' placement with 3 controllers. 143
8.7 Ratios of observed average per-switch FTOs. 147