UNIVERSITY
OF SUSSEX

Understanding the Characteristics of Internet Traffic and Designing an Efficient RaptorQ-based Data Transport Protocol for Modern Data Centres

Submitted for the degree of Doctor of Philosophy
School of Engineering and Informatics, University of Sussex

Prepared by:

Mohammed Alasmar

Supervisor:

Dr. George Parisis

September 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

This thesis conforms to a ‘papers style’ format in which a substantial part of the contents are reproductions from published and un-published submissions to conferences and journals. In particular chapters 3 , 4 and 5 contain publications published/submitted to peer reviewed conferences/journals of which I was the first author. I hereby declare that substantially all of the experimental and theoretical work presented in those papers was my own, except as detailed in the preamble to the papers in those chapters.

Mohammed Alasmar

. September 2019

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. George Parisis. I don't know where to start and how to thank him. My work would not have been possible without his constant guidance, his unwavering encouragement, his many insights and ideas, and his exceptional resourcefulness. And most importantly, his friendship. I have been very fortunate to have him as my supervisor. For all of this, George, thank you!

Special thanks to my second supervisor, Prof. Ian Wakeman for his great support, guidance and supervision. I have also been fortunate to have many fantastic collaborators outside Sussex. I would like to thank all of them: Prof. Jon Crowcroft, Dr. Richard Clegg and Dr. Nickolay Zakhleniuk.

My PhD has been fully funded by the Department of Informatics at the University of Sussex for which I am very grateful.

I also would like to thank Dr. Patrick Holroyd and Dr. Phil Watten for providing us access to computational resources to run large-scale simulations.

Moreover, I would like to extend my thanks to the members of the "Foundations of Software Systems" research group for their outstanding support and friendship throughout my PhD time.

Last but not least, I would like to thank my family, especially my parents, for their unwavering support during my studies. This PhD is devoted to my uncle 'Mohammed' and my aunt 'Samira' who raised me, loved me and encouraged me to pursue my education.

Abstract

This thesis is the amalgamation of research on efficient data transport protocols for data centres and a comprehensive and systematic study of Internet traffic, which came as a result of the need to understand traffic patterns and workloads in modern computer networks.

The first part of the thesis is on the development of efficient data transport protocols for data centres. We study modern data transport protocols for data centres through large scale simulations using the OMNeT++ simulator. We developed and experimented with an OMNeT++ model of NDP. This has led to the identification of limitations of the state of the art and the formulation of research questions with respect to data transport protocols for modern data centres. The developed model includes an implementation of a Fat-tree topology and per-packet ECMP load balancing. We discuss how we integrated the model with the INET Framework and validated it by running various experiments that test different model parameters and components. This work revealed limitations of NDP with respect to efficient one-to-many and many-to-one communication in data centres, which led to the development of SCDP, a novel and general-purpose data transport protocol for data centres that, in contrast to all other protocols proposed to date, natively supports one-to-many and many-to-one data communication, which is extremely common in modern data centres. SCDP does so without compromising on efficiency for short and long unicast flows. SCDP achieves this by integrating RaptorQ codes with receiver-driven data transport, in-network packet trimming and Multi-Level Feedback Queuing (MLFQ); (1) RaptorQ codes enable efficient one-to-many and many-to-one data transport; (2) on top of RaptorQ codes, receiver- driven flow control, in combination with in-network packet trimming, enable efficient usage of network resources as well as multi-path transport and packet spraying for all transport modes. Incast and Outcast are eliminated; (3) the systematic nature of RaptorQ codes, in combination with MLFQ, enable fast, decoding-free completion of short flows. We extensively evaluated SCDP in a wide range of simulated scenarios with realistic data centre workloads. For one-to-many and many-to-one transport sessions, SCDP performs significantly better than NDP. For short and long unicast flows, SCDP performs equally well or better compared to NDP.

In the second part of the thesis, we extensively study Internet traffic. Getting good statistical models of traffic on network links is a well-known, often-studied problem. A lot of attention has been given to correlation patterns and flow duration. The distribution of the amount of traffic per unit time is an equally important but less studied problem. We study a large number of traffic traces from many different networks including academic, commercial and residential networks using state-of-the-art statistical techniques. We show that the log-normal distribution is a better fit than the Gaussian distribution. We also investigate a second, heavy-tailed distribution and show that its performance is better than Gaussian but worse than log-normal. We examine anomalous traces which are a poor fit for all tested distributions and show that this is often due to traffic outages or links that hit maximum capacity. Stationarity tests showed that the traffic is stationary at some range of aggregation times. We demonstrate the utility of the log-normal distribution in two contexts: predicting the proportion of time traffic will exceed a given level (for link capacity estimation) and predicting 95th percentile pricing. We also show the log-normal distribution is a better predictor than Gaussian or Weibull distributions.

Context Statement

This thesis has been prepared as a series of papers for publication, with the exception of Chapters 1, 2 and 6, which serve as the introduction, background and conclusion chapters respectively. The main manuscript that makes up Chapter 3 has been submitted for publication and it is currently under review. Chapter 3 also includes our published and peer reviewed poster paper that includes our early experimentation. The manuscripts that make up Chapter 4 and 5 have been accepted for publication in peer reviewed conferences.

Co-Authors. The papers that make up Chapters 3 of my thesis have three authors [9, 10]: myself, Dr George Parisis[1], who is my main PhD supervisor, and Professor Jon Crowcroft (University of Cambridge)[2]. The paper that makes up Chapter 4 has two authors [7]: myself and Dr George Parisis. Finally, the paper presented in Chapter 5 has four authors [8]: myself, Dr George Parisis, Dr Richard Clegg (Queen Mary University of London)[3] and Dr Nickolay Zakhleniuk (University of Essex)[4].

Context Statement. I am the lead author on all papers. The work outlined in each paper is substantially my own. I designed the solutions, collected the data, implemented the required models, conducted the experiments, evaluated the proposed approaches and wrote up the first draft of each paper.

Contribution form Co-Authors. I received comments from all other authors on the first drafts of each paper, which I incorporated into the final versions. The final version of each paper was reviewed again by my main supervisor. The research questions and general research have been evolved by the tremendous help of my main supervisor through our discussions in the weekly supervisory meetings. Here I provide brief details of the contributions from each co-author to each paper.

Chapter 3: Paper 1&2 [9, 10]. The idea of employing fountain codes for data transport in data centres was part of a workshop paper [65] that was published by my supervisor and Prof. Crowcroft. The resulted papers for this thesis are the full realisation of the research hypothesis that fountain coding can be the foundation for

[1]https://www.sussex.ac.uk/profiles/334868
[2]https://www.cl.cam.ac.uk/~jac22/
[3]http://www.richardclegg.org/
[4]https://www.essex.ac.uk/people/zakhl71400/nick-zakhleniuk

an efficient transport protocol for data centres. My supervisor provided suggestions and feedback for all key aspects of the research, including experimentation. He also helped in formulating the motivation of this work by directing me to the literature and related work on this topic.

Chapter 4: Paper 3 [7]. I was encouraged by my supervisor to build an OMNeT++ model for data centre transport protocols. The source code for all simulation models is my own work. I wrote up the first draft of this paper and I received comments from my supervisor (co-author) on the first draft, which I incorporated into the final version.

Chapter 5: Paper 4 [8]. The idea of studying the characteristics of Internet traffic was inspired by Nickolay Zakhleniuk, who contributed to this paper by highlighting the limitations of some current approaches that study Internet traffic volumes. Nickolay also helped in improving the quality of the paper through his general comments on the text and suggestions in adding more tests to evaluate our approach. The statistical approach [44] that I use in this paper was suggested by Richard Clegg, who also gave valuable feedback to improve the paper through several meetings and discussions. Richard also helped in formulating the research question in this paper and he provided useful comments on the presentation of the experiments. My supervisor contributed in shaping the research motivation and contributions of this paper and in improving the text in this paper.

Contents

List of Figures

Chapter 1

Introduction

Modern computer networks are diverse and complex and support a wide range of network applications with different requirements. Examples of said complexity and diversity include ultra-fast, low-latency data centre networks [17, 72, 138, 176], computing resources from cloud providers (e.g., Amazon Web Services[1], Google Cloud Platform[2], or Microsoft Azure[3]), densely connected satellite networks [91, 28, 57], interconnected ISP networks [32, 31, 150] and various kinds of wireless networks [29, 161, 185]. Efficiency in exchanging data is a key requirement for all these networks and data transport protocols are key in achieving it. At the same time, designing correct and efficient protocols requires understanding the properties of networks, including the traffic they carry, its characteristics and how these change over time.

This thesis primarily looks into the design of efficient data transport protocol for data centres which is extremely important given the rapid changes in the underlying network topologies, traffic workloads and application requirements. As part of understanding traffic workloads and application requirements, we also study the characteristics of Internet traffic, namely the traffic volume, by applying modern statistical analysis toolsets on a large number of publicly available traces.

1.1 Data Transport Protocols for Data Centres

Data centres support the provision of core Internet services such as search (e.g. Google), social networking (e.g. Facebook), cloud services (e.g. Amazon EC2) and video streaming (e.g. NetFlix). Therefore, it is crucial to have in place data transport mechanisms that ensure high performance for the diverse set of supported services. Data centres consist of a large number of commodity servers and switches, support multiple paths among servers, which can be multi-homed, very large aggregate bandwidth and very low latency communication with shallow buffers at the switches.

[1]https://aws.amazon.com/
[2]https://cloud.google.com/
[3]https://azure.microsoft.com/en-gb/

TCP limitations in DCNs. In TCP, packet losses are detected by either triple duplicate ACKs or via retransmission timeouts (RTO). Although triple duplicate ACKs trigger fast retransmission, the recovery delay is still high as at least three packets after the loss need to be received. TCP reacts by halving its congestion window in fast retransmission or resetting it to initial value upon timeout [94]. Obviously, this behaviour has a significant impact on bandwidth utilisation as filling up the available bandwidth will take a long time after a loss.

Furthermore, the RTO is generally in order of hundreds of milliseconds but the RTT in a DCN is often just hundreds of microseconds (i.e. $RTO >> RTT$). This means that retransmission based on the RTO is not appropriate for low-latency DCNs. Improving the quality of user experience of latency-sensitive applications requires an efficient data transport mechanism. For instance, Amazon estimates that every 100 ms increase in latency cuts profits by 1% [74]. Furthermore, in TCP, when a packet is lost or delayed, the subsequent packets have to be buffered in the receiving buffer until the missing packets arrive to fill the gaps. Once the receiving buffer is full, a zero window is advertised to the sender and the sender stops sending even if the link is not fully occupied [16]. This leads to throughput collapse and causes more delay in the network.

One-to-many and many-to-one communication. A significant portion of data traffic in modern data centres is produced by applications and services that replicate data for resilience purposes. For example, distributed storage systems, such as GFS/HDFS [70, 21] and Ceph [192], replicate data blocks across the data centre (with or without daisy chaining[4]). Partition-aggregate [56, 200], streaming telemetry [170, 138, 129], and distributed messaging [4, 101] applications also produce similar traffic workloads. Multicast has already been deployed in data centres[5] and, with the advent of P4, scalable multicasting is becoming practical [172]. As a result, much research on scalable network-layer multicasting in data centres has recently emerged [132, 64, 116, 47, 117].

Existing data centre transport protocols are suboptimal in terms of network and server utilisation for these workloads. One-to-many data transport is implemented through multi-unicasting or daisy chaining for distributed storage [21]. As a result, copies of the same data are transmitted multiple times, wasting network bandwidth and creating hotspots that severely hurt the performance of short, latency-sensitive flows.

In many application scenarios, multiple copies of the same data can be found in the network at the same time (e.g. in replicated distributed storage) but only one replica server is used to fetch it. Fetching data from all servers, in parallel, from all available replica servers (many-to-one data transport) would provide significant benefits in terms of eliminating hotspots and naturally balancing load among servers. However, this is not possible with any of the existing DCN transport protocols. Unfortunately,the state of the

[4] https://patents.google.com/patent/US20140215257

[5] e.g. https://www.rackspace.com/en-gb/cloud/networks

art data transport protocols for data centres (such as NDP [66]) cannot efficiently support such commonly used traffic patterns.

Long and short flows. Modern cloud applications commonly have strict latency requirements [13, 141, 198, 122, 140, 15]. At the same time, background services require high network utilisation [109, 63, 59, 6]. A plethora of mechanisms and protocols have been proposed to provide efficient access to network resources of data centre applications, by exploiting support for multiple equal-cost paths between any two servers [157, 66, 59] and hardware capable of low latency communication [140, 37, 80] and eliminating Incast [38, 39, 202] and Outcast [152]. Recent proposals commonly focus on a single dimension of the otherwise complex problem space; e.g. TIMELY [164], DCQCN [203], QJUMP [84] and RDMA over Converged Ethernet v2 [98] focus on low latency communication but do not support multi-path routing. Other approaches [6, 59] do provide excellent performance for long flows but perform poorly for short flows [157, 109]. None of these protocols supports efficient one-to-many and many-to-one communications.

TCP Incast. Incast [38] is a catastrophic TCP throughput collapse which occurs when a large number of synchronised short flows hit the same switch queue in the data centre. In distributed storage or partition/aggregate workloads Incast occurs at the queue of the switch port connected to the storage client or aggregator, respectively. Incast occurs when, under severe congestion, packet loss is high and so TCP falls back to retransmission timeouts (RTOs) to recover. Several techniques have been proposed to mitigate Incast [13, 66], but none of these approaches can efficiently support one-to-many and many-to-one workloads nor multi-path communication or multi-homed network topologies.

Multi-path data transport. TCP or other single-path variations of TCP for DCNs rely on ECMP [96] to distribute multiple flows across all available paths. However, per-flow ECMP can cause flow collisions which lead in hotspots and significant performance deterioration. MPTCP [63] improves resource utilisation between two endpoints by using multiple paths. MPTCP also relies on ECMP to distribute data over multiple paths and therefore suffers from sub-flow collisions [66]. Moreover, MPTCP performance suffers when the quality of the used paths varies significantly (e.g. due to persistent or transient hotspots in the network due to link failures or bursty traffic). Per-packet load balancing across different paths is also problematic in both TCP and MPTCP because out-of-order packets significantly hurt the overall performance.

The above discussion leads us to formulate the following research question:

- **Research Question 1.** *How can we design a data transport protocol that supports diverse and modern communication patterns in data centres, while providing high* *goodput, low completion times and high network utilisation to a diverse set of data centre applications?*

1.2 Internet Traffic Characterisation

Internet traffic characterisation is an important problem for network researchers and operators. The subject has a long history. Early research [154, 46] resulted in discovering that the correlation structure of network traffic exhibits self-similarity and that the duration of individual flows of packets exhibit heavy-tails [69]. By comparison, the distribution of the volume of traffic seen on a link in a given time period has been comparatively less studied. This is surprising as this quantity can be extremely useful in network planning. Several studies showed that network traffic follow a Gaussian distribution [134, 52, 53]. All these studies are based on straightforward goodness-of-fit tests (e.g., Quantile-Quantile (Q-Q) plots) and relevant correlation tests that are used to assess how well captured traffic traces are fitted to Gaussian or heavy-tailed distributions [60, 120, 143, 126]. As discussed in [44], these statistical approaches can produce a substantially inaccurate assessments about whether traffic volume samples follow a Gaussian/heavy-tailed distribution or not. This is because they are very sensitive to the volatile tail of said distributions and, therefore, require a very large sample of data to produce reliable results [44].

Traffic modelling has significant practical value; e.g. (1) in bandwidth provisioning which is one of the key activities of network management that aims at achieving desired levels of quality of service. Current practices of provisioning by network operators are based on rules-of-thumb and on rough traffic measurements. For example, the link provisioning is done by adding to the average obtained via SNMP a safety margin (e.g., 30% of the calculated average) [42]. However, traffic measurements showed that traffic perform high variations at small aggregation times [127, 3, 134]. Consequently, the link usage and the bandwidth need are not correctly estimated, resulting in poor network performance (in the case of underestimation), respectively waste of resources (in the case of overestimation). In general, the key challenge in bandwidth provisioning is to create an adequate model of current and future traffic behaviour. (2) A traffic model can be used to predict customers' bills using the 95th percentile method. This method is based on measuring the utilisation of a customer link in 5-minute bins throughout a month, and then computing the 95th percentile of these values as the billing volume [180, 160].

Hence, we formulate the second research question as follows:

- **Research Question 2.** *i. How can we understand the Internet traffic behaviour by investigating real Internet traces? ii. Can we find a well-established statistical methodology that can be used to find the best model that can characterise traffic volume? iii. What are the benefits of characterising and modelling Internet traffic volumes?*

1.3 Research Contribution

This thesis is the amalgamation of research on efficient data transport protocols for data centres and a comprehensive and systematic study of Internet traffic, which came as a result of the need to understand traffic patterns and workloads in modern computer networks. More specifically, the research contribution of this thesis is as follows.

Contribution 1. We propose SCDP [10] (Research Question 1), a general-purpose transport protocol for data centres that, unlike any other protocol proposed to date, supports efficient one-to-many and many-to-one communication. This, in turn, results in significantly better overall network utilisation, minimising hotspots and providing more resources to long and short unicast flows. At the same time, SCDP supports fast completion of latency-sensitive flows and consistently high-bandwidth communication for long flows. SCDP eliminates Incast [38, 39, 202] and Outcast [152]. All these are made possible by integrating RaptorQ codes [67, 173] with receiver-driven data transport [66, 140], in-network packet trimming [40, 66] and Multi-Level Feedback Queuing (MLFQ) [24].

RaptorQ codes are systematic and rateless, induce minimal network overhead and support excellent encoding/decoding performance with low memory footprint. They naturally enable one-to-many and many-to-one data transport. They support per-packet (encoded symbol) multi-path routing and multi-homed network topologies [85, 178]; packet reordering does not affect SCDP's performance, in contrast to protocols like [109, 13, 157]. In combination with receiver-driven flow control, and packet trimming, SCDP eliminates Incast and Outcast, playing well with switches' shallow buffers. The systematic nature of RaptorQ codes enables fast, decoding-free completion of latency-sensitive flows by prioritising newly established ones, therefore eliminating loss (except under very heavy loads). Long flows are latency-insensitive so lost symbols can be recovered by repair ones; SCDP employs pipelining of source blocks, which alleviates the decoding overhead for large data blocks and maximises application goodput. SCDP is a simple-to-tune protocol, which, as with NDP and scalable multicasting, will be deployable when P4 switches [30] are deployed in data centres.

We found that SCDP improves goodput performance by up to ∼50% compared to NDP with different application workloads involving one-to-many and many-to-one communication. Equally importantly, it reduces the average FCT for short flows by up to ∼45% compared to NDP under two realistic data centre traffic workloads. For short flows, decoding latency is minimised by the combination of the systematic nature of RaptorQ codes and MLFQ; even in a 70% loaded network, decoding was needed for only 9.6% of short flows. This percentage was less than 1% in a 50% congested network. The network overhead induced by RaptorQ codes is negligible compared to the benefits of supporting one-to-many and many-to-one communication. Only 1% network overhead was introduced under a heavily congested network. RaptorQ codes have been shown to perform exceptionally well even on a single core, in terms of encoding/decoding rates. We, therefore, expect that with hardware offloading, in combination with SCDP's block pipelining mechanism, the required computational overhead will be insignificant.

Contribution 2. We develop a simulation framework [7] for data centre transport protocols (Research Question 1), including a simulation model for NDP [66], the state of the art data transport protocol for data centres, a fat-tree network topology and per-packet ECMP load balancing. We describe how we integrated our model with the INET Framework and present example simulations to showcase the workings of the developed framework. For that, we conducted large scale experimentation evaluating and validating different components and parameters of the developed models. We have used the developed framework to understand the behaviour of NDP in various diverse network scenarios, including one-to-many and many-to-one communication, Incast and Outcast, for evaluating SCDP and comparing its performance with NDP.

Contribution 3. We present an extensive statistical analysis applied to the real Internet traffic datasets (Research Question 2) [8]. We use a rigorous statistical approach to fitting a statistical distribution to the amount of traffic within a given time period. We investigate the distribution of the traffic over a wide range of values of timescales. We show that this distribution has considerable implications for network planning; for assessing how often a link is over capacity and in particular for service level agreements (SLAs), and for traffic pricing, particularly using the 95th percentile scheme [180]. Previous authors have claimed that the normal (or Gaussian) distribution is the best fit for the traffic volume [134, 54, 52]. Others claim that heavy-tailed distributions are better in fitting the traffic [51, 199].

We use a well-established statistical methodology [44] to show that a log-normal distribution is a better fit than Gaussian or Weibull for the vast majority of traces. This holds over a wide range of timescales T (from 5 ms to 5 sec). We study a large number of publicly available traces from a diverse set of locations (including commercial, academic and residential networks) with different link speeds and spanning the last 15 years. We develop the mathematics to show how often a link following a given distribution will be over a given capacity and show that our approach improves greatly on results assuming traffic follows a normal distribution. We further show that if an ISP wishes to estimate future transit bills that use the 95th percentile billing scheme, then the log-normal is a better model than the normal distribution.

Overall. The techniques that we suggest to study Internet traffic could be adopted for data centres traffic. Besides, understanding the characteristics of Internet traffic by investigating real Internet traces from different networks is a right step towards designing and developing new protocols. Hence, *Contributions 1*&2 of designing data transport protocol for data centres are achieved by the help of the outcomes in *Contribution 3*.

1.4 Thesis Structure

This thesis is structured as follows:

- **Chapter 2** gives an overview of the background literature that is related to the work presented in this thesis. The chapter first introduces the most commonly deployed data centre topologies and discusses modern data centre applications and produced traffic workloads. Next, it describes recently proposed TCP variants and other data transport schemes in DCNs. Additionally, it provides a brief description of fountain codes, including LT and RaptorQ codes. We then focus on literature related to the characterisation of the volume of Internet traffic.

- **Chapter 3** includes the paper on developing SCDP [10], a fountain coding-based data transport protocol. Through large-scale simulations, we demonstrate the superiority of SCDP in comparison to TCP and NDP for a large and diverse set of traffic workloads and networking scenarios. For completeness, we also provide our early publication that set the foundation for researching SCDP [9].

- **Chapter 4** includes the paper on developing a simulation framework for data centres and evaluating data centre protocols in OMNeT++ [7]. The paper describes the developed framework, including the fat-tree topology, per-packet and per-flow ECMP, flow scheduling and the NDP model. The paper then evaluates the performance of NDP in OMNeT++ in a simulated fat-tree topology.

- **Chapter 5** includes the paper on the characterisation of Internet traffic volumes using a robust statistical approach [8]. The findings suggest that the log-normal model is a good distribution to fit Internet traffic, which is important for operators for network dimensioning and traffic billing purposes.

- **Chapter 6** draws conclusions arising from the work of this thesis, and presents directions for future work.

1.5 Related publications

The works described in this thesis is included in the following published and submitted papers.

- Mohammed Alasmar, George Parisis and Jon Crowcroft, “Polyraptor: embracing path and data redundancy in data centres for efficient data transport”. In Proceedings of ACM SIGCOMM 2018 (Poster Sessions), Budapest, Hungary [9].

- Mohammed Alasmar, George Parisis and Jon Crowcroft, “SCDP: Systematic Rateless Coding for Efficient Data Transport in Data Centres”. IEEE/ACM Transactions on Networking 2019 (submitted)[6].

- Mohammed Alasmar and George Parisis, “Evaluating Modern Data Centre Transport Protocols in OMNeT++/INET”. In Proceedings of the OMNeT++ Community Summit 2019, Hamburg, Germany [7].
- Mohammed Alasmar, George Parisis, Richard Clegg and Nickolay Zakhleniuk, “On the Distribution of Traffic Volumes in the Internet and its Implications”. In Proceedings of IEEE INFOCOM 2019, Paris, France [8].

[6]https://arxiv.org/abs/1909.08928

Chapter 2

Background

In this chapter, we present research and background works that are relevant to this thesis. The background includes two main strands of work as follows:

- In the first part of this chapter, we present the background that is related to designing a reliable data transport protocol in data centre networks. This is divided into these sections. Firstly, we present the network architecture of today's data centres (§2.1). Secondly, we describe mechanisms for load balancing and multicasting in data centres (§2.2). Thirdly, we discuss the main TCP limitations in today's data centres (§2.3). Next, we provide an overview of the most common and recent data transport protocols and flow scheduling approaches for DCNs (§2.4, §2.5). Besides, we discuss research on data centre network traffic characteristics (§2.6). Furthermore, we motivate the benefits of adopting fountain codes in data transports by discussing their properties (§2.7). We also introduce the used network simulator in this work (§2.8).

- In the second part of this chapter, we present the background of Internet traffic volume characteristics and how to use a robust statistical approach to model traffic volumes. We also describe some use cases of such model (§2.9).

2.1 Data Centre Network Topologies

2.1.1 Switch-centric and server-centric topologies

Data Centre Network (DCN) topologies are categorised into server-centric and switch-centric according to the employed routing protocols and the components that perform routing and switching. In server-centric topologies, servers act as the relaying nodes in multi-hop networks. BCube [85] and DCell [87] are examples of this approach. On the contrary, in switch-centric topologies, switches act as the relaying nodes. Fat-tree [5], VL2 [83] and PortLand [145] are switch-centric topologies.

Fat-tree [5] is a commonly deployed topology in modern data centres. A fat-tree data

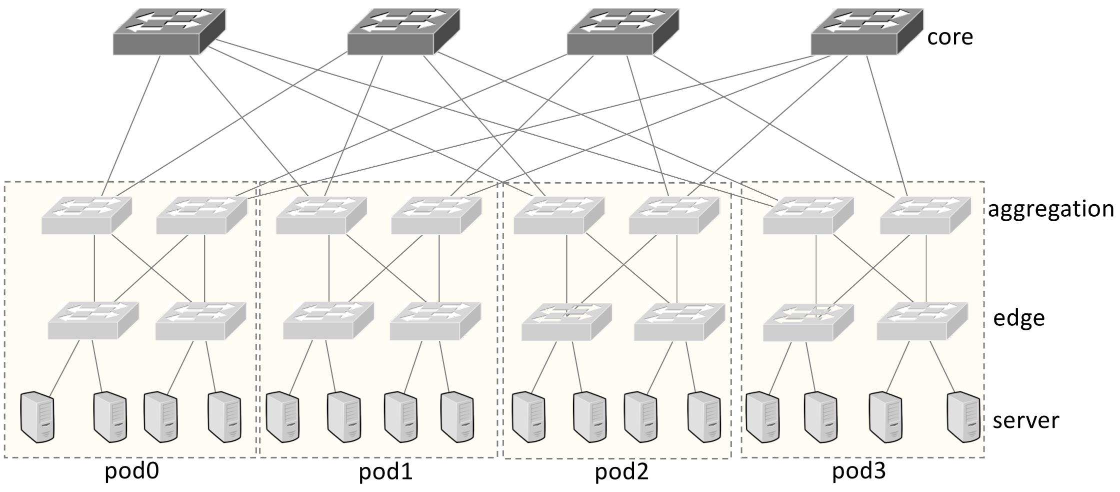


Figure 2.1: Fat-tree topology [5]

centre network is divided into three layers: the core, aggregation, and edge layer. It is a folded Clos topology [1]. An important feature of this network topology is its ability to offer full bisection bandwidth. Full bisection bandwidth means that any server can communicate with any unused server at full line-rate. Non-blocking networks provide full bisection bandwidth. The edge of the network is usually oversubscribed [5].

A Fat-tree DCN is composed of k pods. Each pod contains k servers and k switches. These switches are divided into two layers, each one consisting of $k/2$ switches. The first layer is the edge (top of rack switch) where each switch is connected to $k/2$ of the servers in the same pod, while the second layer is the aggregation layer where each switch is connected to $k/2$ of the core switches. Each core switch is connected to one aggregation switch of each pod. The maximum number of servers in a Fat-tree with k pods is $(k/3)^4$. In a k-ary Fat-tree topology there are $k/2^2$ core switches which is the same number as the shortest-paths between any two servers on connected pods. Figure 2.1 shows a *4*-ary Fat-tree topology consisting of 16 hosts.

In fat-tree topologies, servers can communicate with other servers in the same rack at the full rate of their interfaces (1:1 over-subscription). However, the presence of multiple levels leads to high oversubscribed ratios at the aggregation and core switches, for example, 1:20 oversubscribed indicates that 1 Gbps of up-link for 20 servers each one with 1 Gbps interface.

The existence of multiple equal-cost paths in this topology can be exploited to deploy multipath protocols (i.e., MPTCP [158]), which provide better performance and higher throughput.

Fat-tree topologies have a number of limitations. Most importantly, they suffer from wiring complexity and high deployment costs, especially as the number of servers increases. In case of failure, there are no direct alternative nodes around the failure node. This means that only the source switch can be used to reroute the failure. This indicates that the nodes that have alternative paths are on the other side of the network and comes from the fact that fat-tree networks are symmetric. The AB fat-tree topology has been proposed to provide better network-level failure recovery [118]. Moreover, it is hard for fat-tree topologies to be expanded by a small number of hosts since any expansion requires a full pod to be added to maintain consistency in the network.

PortLand [145] applies a logically centralised Fabric Manager that contains the information of the network configuration such as topology. The fabric manager is a user process running on a specific machine inside the data centre. The idea of deploying a centralised manager can help in reducing the complexity of routing, but at the same time it decreases robustness. Therefore, the amount of information in the fabric manager should be limited to maintain high robustness. In Virtual Layer 2 (VL2) topology [83] the key objective is to redesign the traditional Layer 2/3 network architectures to be more efficient for modern data centres i.e., VL2 requires modifying end-host OS and network stacks. DCell [87] and BCube [85] are server-centric DCN topologies, where servers act as relaying nodes in multi-hop networks. **Jellyfish** [178] is an unstructured data centre network topology, where it is easy to add/replace servers and switches.

Our proposed transport protocol [10] is agnostic to the underlying data centre network topology and only requires specific in-network functionality (i.e. packet trimming) that is or will be readily available when P4 [30] switches are deployed in data centres.

2.1.2 Google, Facebook and Microsoft DCNs

Google [176], Microsoft [177, 83] and Facebook [17] data centre network topologies follow the fat-tree design. In the Jupiter [176] data centre generation (shown in Figure 2.2), Google use their own switches using 16 $\times$ 40Gbps merchant silicon and a ToR switch has 48 $\times$ 40Gbps connections to servers in the rack and 16 $\times$ 40Gbps connections to the aggregation switches. Four such switches form a Middle Block (MB), which serves as a building block in the aggregation block. The logical topology of an MB is a 2-stage blocking network, with 256 $\times$ 10Gbps connections to ToRs and 64 $\times$ 40Gbps connections to the spine. Each ToR connects to eight MBs with dual redundant 10Gbps links. An aggregation block has 512 $\times$ 40Gbps or 256 $\times$ 40Gbps links towards the spine blocks. A spine block has six switches with 128Gbps ports to the aggregation blocks. There are 64 aggregation blocks.

2.1.3 Flexibility and performance optimisation in modern data centres

Recently, a number of techniques have been proposed to give operators more room for performance optimisations. This includes some new technologies that are deployable in data centres such as P4 Switches [30], Advanced NICs [73], DPDK [171] and RDMA [98].

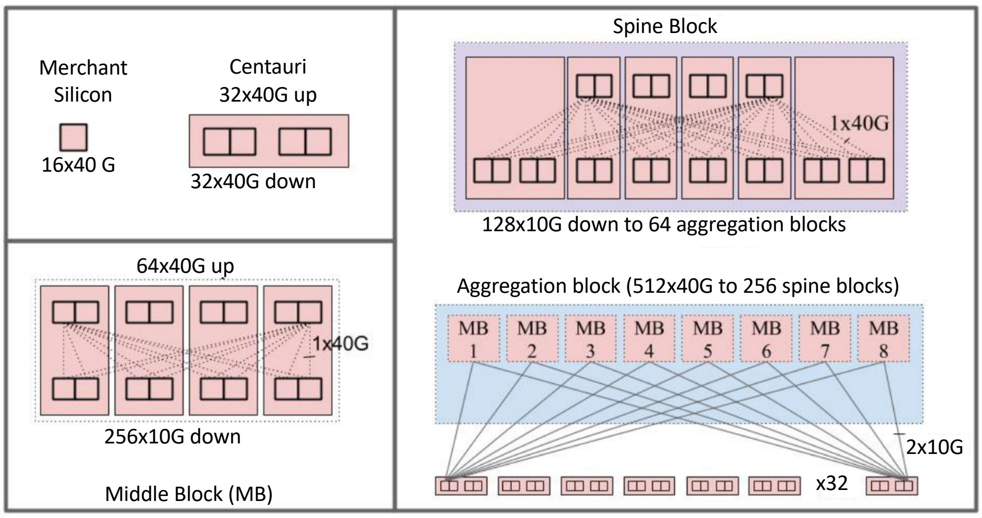


Figure 2.2: Google Jupiter data centre topology [176]

P4 Switches. Programmable data planes (PDPs) allow the packet processing functions to be changed at the forwarding devices, i.e., switches can apply new forwarding functions at the line rate. This is orthogonal to programmable forwarding planes (e.g., SDN) where different forwarding rules can be selectively applied to packets. PDPs make it easier to deploy traffic control schemes that depend on custom in-network processing or feedback. For example [14, 15, 95, 66] rely on custom packet headers and feedback from switches. P4 (Programming Protocol-Independent Packet Processors)[1] [30] is a high-level open-source language to program Protocol-Independent Switch Architecture (PISA) switches which is vendor independent, and protocol independent (i.e., operates directly on header bits and can be configured to work with any higher layer protocol). P4 compiler can also compile P4 code to run on a general purpose processor as software switches [146].

Advanced NICs. Several vendors have been developing NICs with advanced offloading features. These features allow complex operations at high line rates of data centres (40 Gbps and more) which at the OS may incur significant CPU overhead and additional communication latency. Examples of offloading features include cryptography, quality of service, encapsulation, congestion control, storage acceleration, erasure coding, and network policy enforcement. Examples of these advanced NICs: Mellanox ConnectX [135] and Microsoft SmartNIC [73].

DPDK. Data Plane Development Kit (DPDK) [171] consists of libraries to accelerate packet processing by allowing userspace programs to directly access NIC buffers to read incoming packets or write packets for transmission. It works by bypassing Operating System's networking stack and use polling instead of interrupts. DPDK is able to reduce the number of required cycles to process a packet by up to 20x on average [62, 146].

[1] `https://p4.org/specs/`

RDMA. Remote Direct Memory Access (RDMA) [86, 203, 98, 175] is a transport protocol that allows delivery of data from one machine to another machine without involving the OS networking protocol stack. RDMA operates on the NICs of machines communicating. Compared to TCP, RDMA offers higher bandwidth and lower latency at lower CPU utilization. TCP has been the dominant transport protocol across data centres for the majority of applications. Since implemented as part of OS protocol stack, using TCP at high transmission rates can exhaust considerable CPU resources and impose a notable amount of communication latency. Unlike TCP, RDMA needs a lossless network; i.e. there must be no packet loss due to buffer overflow at the switches. RDMA over Converged Ethernet (RoCEv2) [98] allows RDMA over an Ethernet network by using PFC (Priority-based Flow Control) [149]. PFC prevents buffer overflow by pausing the upstream sending entity when buffer occupancy exceeds a specified threshold.

2.2 Load Balancing and Multicasting in Data Centres

Load Balancing in data centre networks. Routing protocols in data centre networks, such as Equal-Cost Multi-Path (ECMP) [96], which provide per-flow load balancing can cause significant underutilisation in the network due to collisions of large flows [59, 66]. Randomised load balancing by switching any incoming flow randomly to one of the available links can cause high congestion in some links. Per-packet ECMP may cause out-of-order packet delivery, which affects the performance of all flows when TCP is employed as a transport protocol [66].

In [59], the authors showed that random packet spraying (RPS) leads to better load balance and network utilisation than per-flow ECMP. RPS incurs little packet reordering since it exploits the symmetry in DCNs. This approach is less complex than other approaches as Hedera [6] and MPTCP [157] and it is readily implementable. RPS would be a good load balancing option to work with transport protocols that can handle out-of-order packets as NDP [66] and SCDP [10], the transport protocol proposed in this thesis.

Multicast support in data centre networks. In modern data centres, one-to-many group communication workloads are common for tasks, such as file dissemination, data replication, distributed messaging etc. [88, 172, 117]. One-to-many workloads exploit support for network-layer multicast (e.g. with [172, 132, 64, 116, 68]) and coordination at the application layer; for example, in a distributed storage scenario, multicast groups could be pre-established for different replica server groups or setup on demand by a metadata storage server [147]. Multicast is widely considered as the best solution since it natively involves less traffic load and server overheads [172]. With recent advances in scalable data centre multicasting, a very large number of multicast groups can be deployed with manageable overhead in terms of switch state and packet size. For example, Elmo [172] encodes multicast group information inside packets, therefore minimising the need to store state at the network switches. Elmo can support an extremely large number of groups, which can be encoded directly in packets, eliminating any maintenance overhead associated with churn in the multicast state. "In a three-tier data centre topology with 27K hosts, Elmo supports a million multicast groups using a 325-byte packet header, requiring as few as 1.1K multicast group-table entries on average in leaf switches, with a traffic overhead as low as 5% over ideal multicast" [172]. Eventually, multicast has already been deployed in data centres[2], but with the advent of P4 [30], scalable multicasting is becoming practical [172].

2.3 TCP Limitations in Data Centre Networks

TCP: Transmission Control Protocol

The Transmission Control Protocol (TCP) is the de-facto data transport protocol for the Internet, ensuring reliable, in-order data transmission between two endpoints. TCP ensures reliable data delivery by applying error recovery and flow control through an end-to-end connection. TCP handles packet losses, that are due to transmission errors or network congestion. TCP congestion control algorithms (such as Tahoe [16], Reno [16], Vegas [34], New Reno [94], CUBIC [89], and Compound [179]) throttle the sender when the network becomes congested.

TCP is nowadays considered ill-suited for modern DCNs [65, 66, 24] that support high-bandwidth, low-latency links and applications with wildly varying requirements. Known problems that TCP suffers from in such networks include TCP Incast [38] and Outcast [152], latency due to queue build-up and buffer pressure [13] and low resource utilisation [66]. Moreover, TCP cannot inherently support the extremely common one-to-many and many-to-one traffic workloads of modern data centres. These limitations are shown in Figure 2.3, which we discuss in the following sections.

2.3.1 TCP Incast

In many-to-one communication, the presence of many concurrent and parallel flows to a single switch result in throughput collapse. This phenomenon is called TCP Incast [38]. It is a drastic reduction in application level throughput that is presented when simultaneous TCP requests from a number of concurrent senders (see Figure 2.4) are sent to a single receiver. The congestion at the switch buffers leads to extensive packet loss. The retransmission of any lost packet occurs after the RTO (Retransmit TimeOut) timer fires. When there are no massive losses, retransmission can be faster by using duplicate acknowledgements to invoke Fast Retransmit phase. The RTO is generally in order of hundreds of milliseconds but the RTT in a DCN is often just hundreds of microseconds

[2]e.g. https://www.rackspace.com/en-gb/cloud/networks

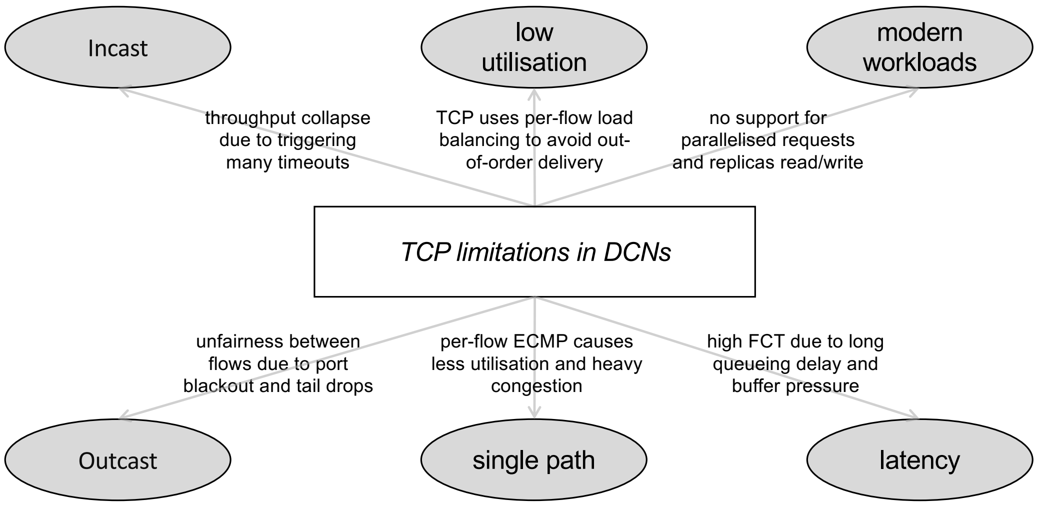

Incast
low utilisation
modern workloads
throughput collapse due to triggering many timeouts
TCP uses per-flow load balancing to avoid out-of-order delivery
no support for parallelised requests and replicas read/write
TCP limitations in DCNs
unfairness between flows due to port blackout and tail drops
per-flow ECMP causes less utilisation and heavy congestion
high FCT due to long queueing delay and buffer pressure
Outcast
single path
latency

Figure 2.3: TCP limitations in DCNs

(i.e. $RTO >> RTT$). This means that the sender should wait a minimum of RTO before receiving any subsequent segments, which leaves the relevant links entirely idle [39, 38].

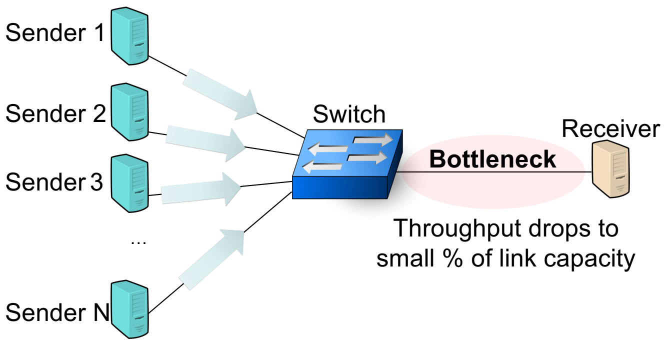


Figure 2.4: Incast phenomenon

This behaviour causes severe reduction in the application throughput and it increases the job completion times that are observed by the users. Employing developed TCP congestion control mechanisms such as NewReno, SACK, and ECN improve the throughput, but do not solve the throughput collapse due to TCP Incast [38]. Approaches that deal with TCP Incast problem will be discussed later in this chapter.

2.3.2 TCP Outcast

Bandwidth sharing via TCP in commodity data centre networks organised in multi-rooted tree topologies can lead to severe unfairness, which is termed as the TCP Outcast problem. TCP Outcast occurs when a small set of flows share a bottleneck with a large set of flows. This is common in data centre tree topologies where any edge switch with two input ports has two input competing flows (a small set of flows from the neighbour servers and large set of flows from servers in different pods) which are forwarded at one common output port, as shown in Figure 2.5. The small set of flows suffers from unfairness at this bottleneck, which is manifested as lower throughput compared to other flows [152].

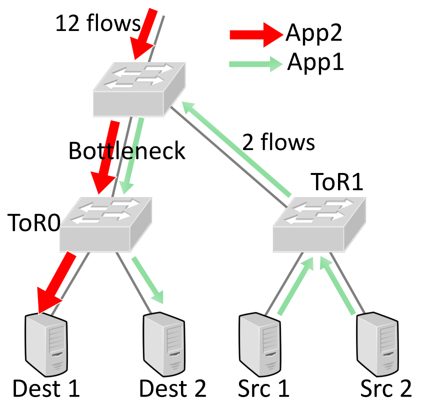


Figure 2.5: Outcast phenomenon [152]

TCP outcast is related to a phenomenon called port blackout. This occurs due to employing tail-drop queues in commodity switches. Port blackout can be essentially explained as follows. When a series of packets enter the switch from different input ports, they must compete for the only output port. Some of the packets may luckily get into the switch's cache, while the rest of them will only be discarded, since the queue in the output port is full. Figure 2.6 illustrates an example of port blackout where there are packets coming in from input port A and packets coming in from input port B and they are competing at output port C. The arrived packets from port A are dropped due to a full queue. The port blackout occurs due to the existence of flows with roughly the same packet size. Furthermore, TCP outcast appears in many-to-one communication forms where two sets of flows compete at the input ports [152].

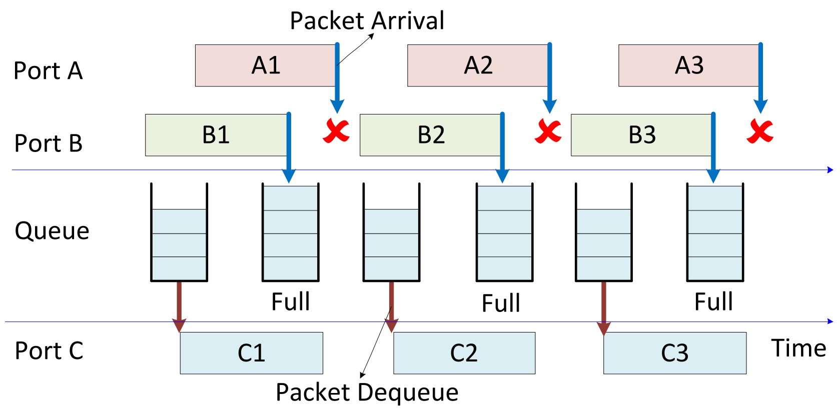


Figure 2.6: Example of Port Blackout [152]

Intuitively, flows with high RTT achieve lower throughput than flows with low RTT. However, the presence of TCP outcast in DCN causes higher throughput for flows that have higher RTT. In [152], the authors have evaluated a number of solutions to overcome the TCP outcast problem. These solutions include using different queuing mechanisms such as Stochastic Fair Queuing (SFQ)[133] and Random Early Detection (RED) [75].

2.3.3 Buffer pressure and queue build-up

In DCNs, there are two types of TCP flows: short lived and long lived, with sizes that typically range from 2 KB to 100MB [201, 13]. Short-lived flows are latency sensitive, while long-lived flows are latency insensitive. The sharing of a bottleneck between short and long flows increases the latency for the short flows. Therefore, short flows may suffer from poor performance due to a large number of packets lost, timeouts and frequent retransmissions. The queue build-up results in full buffer space, since buffers are shared resources. This leaves very little space for any incoming packets. As a consequence of frequent retransmissions due to packet drops, the TCP sender needs to reduce the size of congestion window and needs more RTTs to complete the flow. Furthermore, in data centre networks the majority of the traffic is bursty, and hence, packets of short-lived TCP flows get dropped frequently [13]. Like TCP Incast, buffer pressure occurs due to the presence of many requests to a relatively small switch buffer, but it occurs without requiring barrier-synchronised flows [13].

In a TCP connection with large traffic, queues build up gradually. The queue build-up depends on how much buffer space is not occupied and on the present traffic. When both short and long flows traverse through the same switch, the performance of the short flows is significantly affected. In such scenario, the short flows suffer from high queuing delay with a high probability of packet dropping. Queue build-up can be avoided by employing a proactive data transport mechanism (e.g. DCTCP [13]) which can minimise the queue occupancy by imposing an effective congestion control algorithm [13].

It was found that latency-sensitive TCP connections with packet loss take on average five times longer to complete than those without any loss due to the loss detection and recovery mechanism of TCP [74]. Also, it was reported that most losses occur at the tail of a transmission burst, which does not have enough acknowledgements to trigger fast retransmission; thus, losses are recovered through expensive retransmission timeouts (RTOs) [74, 50].

2.3.4 Single-path TCP (per-flow connection)

In Modern DCNs, transport protocols should exploit the existence of multiple equal-cost paths to better balance traffic in the network, eliminating hotspots and achieving high throughput. TCP and its single-path variants, such as DCTCP, do not fully utilise the available bandwidth in DCN, which results in the degradation of throughput and unfairness. Multi-path TCP (MPTCP) [157] was proposed to overcome the utilisation limitation of TCP.

2.3.5 Modern workloads/applications in DCNs

Applications that commonly run in DCNs can be categorised as follows [3] [151]:

Control and management applications: for example, clock synchronisation (e.g., Precision Time Protocol [181]) and cluster management (e.g., Google's Borg [190]).

Data storage and retrieval: for example, distributed file systems (e.g., HDFS/GFS [70], distributed database systems (e.g., Spanner [45]) and key-value stores (e.g., Memcached [136]). The Hadoop Distributed File System (HDFS) [21] (&Google File System [70]) is designed to reliably store very large files across data nodes in a large cluster (MapReduce [56] is a framework to process in parallel large amount of data stored over the data nodes). It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance (the replication factor is three). **Write operation:** (Figure 2.7a) HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. Clients write data to replicas following a daisy chain and pipelined manner where each replica is forwarded as it is received by each replica server. In GFS's daisy-chained writes, the client writes to the topologically nearest replica, which in turn writes to the next replica, and so on. **Read operation:** (Figure 2.7b) a client reads data from the topologically nearest replica. HDFS read operations happen in parallel (parallel reads of different blocks in the file) instead of sequential like write operations. The read operation ensures parallelism by concurrently requesting to read distinct blocks from the same client.

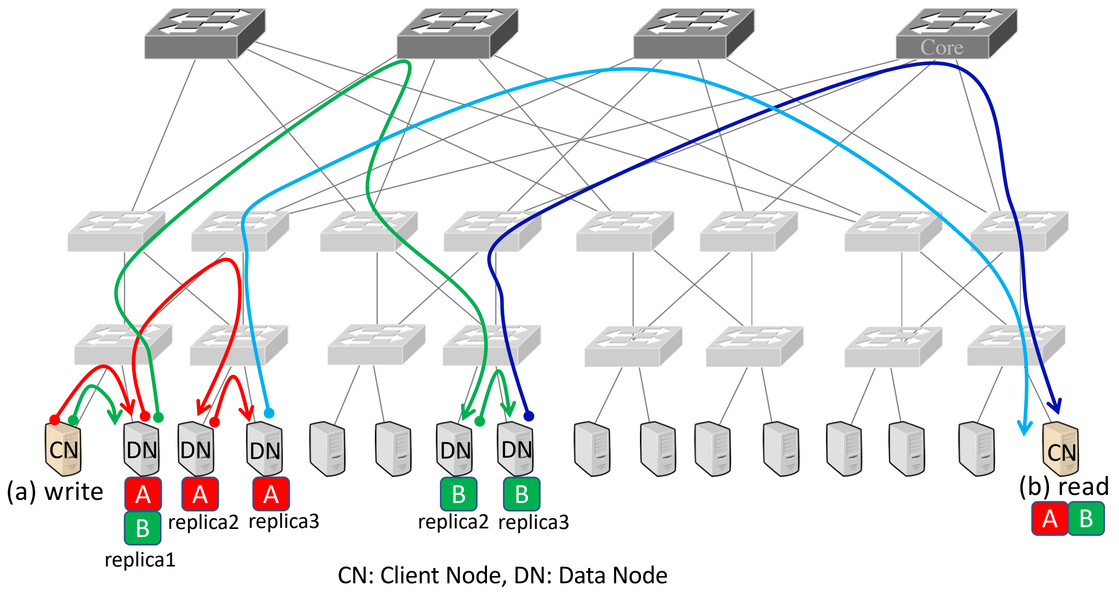


Figure 2.7: Write/read requests in HDFS. Example of writing a 256 MB file (2 blocks 128 MB each) by a client node and reading the file by another client node

[3]Our proposed data transport protocol SCDP supports all these workloads

Applications serving users' needs: for example, data processing frameworks (MapReduce [56]), graph processing (e.g., Apache Giraph [18]), stream processing (e.g., Apache Storm [20]), machine learning analytics (e.g., Tensorflow [2]), Web traffic [19], search engine and social network backends. In addition to these, tenant applications running in VMs, which can be any of the previous applications or custom applications.

In the MapReduce pattern [56], a mapper reads its input from the distributed file system (DFS), performs user-defined computations on the input read and writes intermediate data to the disk. Each reducer pulls the intermediate result from different mappers (shuffle phase), merges them, and writes the output to the DFS, which then replicates it to multiple destinations.

In the partition-aggregation pattern [13], user-facing online services (e.g., search results in Google or Bing, home feeds in Facebook) receive requests from users and send it downward to the workers using an aggregation tree. At each level of the tree, individual requests generate activities in different partitions. Ultimately, worker responses are aggregated and sent back to the user within strict deadlines.

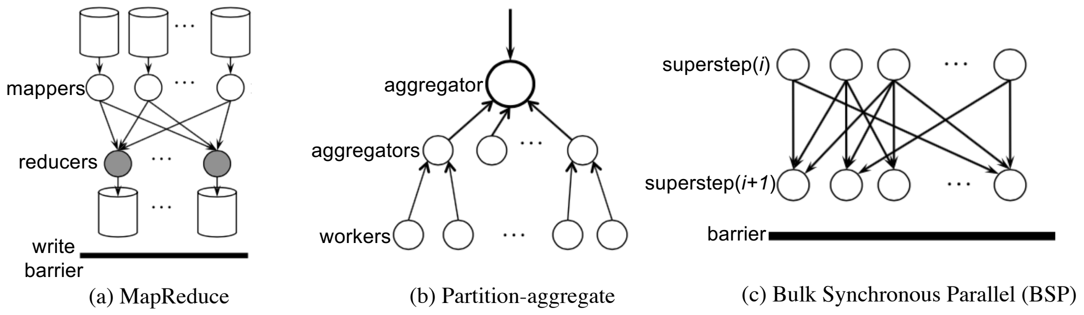


Figure 2.8: Communication patterns examples in DCN [41]

2.4 Existing Data Transport Protocols and Flow Scheduling Approaches for DCNs

Recently, a large body of data transport protocols and flow scheduling approaches aimed at tackling various performance targets in data centres have been proposed. Approaches focus on either achieving low latency such as DCTCP [13], pFABRIC [15], PIAS [24], pHost [80], NDP [66], HOMA [140], D2TCP [186], CAPS [97], TIMELY [164], DCQCN [203] and QJUMP [84] or high throughput as in MPTCP [63], MMPTCP [109], Hedera [6], RPS [59] and FMTCP [48]. NDP [66] and Homa [140] appear to perform well with respect to both low latency and high throughput requirements by combining a number of data transport mechanisms. Incast was discussed in some of these works [189, 13, 195, 102, 140, 66]. In this section, we discuss some of the proposed data transport protocols and briefly discuss how they tackle specific TCP limitations in data centres.

2.4.1 Data Centre TCP (DCTCP)

DCTCP [13] has been proposed to improve the TCP congestion control mechanism for DCNs. It utilizes the Explicit Congestion Notification (ECN) option [162], which is available in modern data centre switches. DCTCP aims to overcome the TCP impairments such as Incast, queue build-up and buffer pressure. DCTCP follows a proactive approach since it attempts to avoid packet loss rather than waiting until packet loss occurs. DCTCP uses an efficient marking scheme mechanism at switches. The switch marks any incoming packet's CE (Congestion Experienced) code point when the queue occupancy is larger than marking threshold K. At the sender, DCTCP responds to a marked ACK (when queue length is larger than threshold K) by reducing the *cwnd* based on the fraction of marked packets in the most recent window of data. The receiver sets the ECN flag when a packet with the CE mark is received. This occurs when the bottleneck link buffer size is larger than a threshold K. For a set of synchronised flows competing for a shared link of capacity C packets/second with identical RTT seconds, the lower bound of the marking threshold K is $C \times RTT/7$ [13].

The experimental results of DCTCP show that short flows achieve low latency and long flows achieve high throughput [13]. TCP convergence and fairness are very poor compared to DCTCP [13]. TCP requires several seconds to converge [114]. This problem is caused by the poor performance of receive buffer tuning [13].

Two drawbacks of DCTCP have been identified in [104]. Firstly, DCTCP and TCP do not coexist well. It is observed that the existence of DCTCP causes regular TCP flows to stop completely [104]. Secondly, DCTCP does not allow ECN Capable Transport (ECT) to be set on neither SYN nor SYN/ACK packets [162]. Therefore, the non-ECT SYN and SYN/ACK packets will be dropped in the congestion case when the queue length is larger than the marking threshold. This reduces the probability of establishing a new DCTCP connection promptly. The authors of [104] have proposed employing ECT in SYN and SYN/ACK packets to improve DCTCP connection establishment. DCTCP does not exploit the existence of multiple equal-cost paths in modern data centres.

2.4.2 Multipath TCP (MPTCP)

In conventional TCP, the flow has to be sent over a single path. This single-path approach does not work efficiently with today's DCNs. In modern data centres, the availability of many paths between two endpoints should be exploited effectively [63]. Multipath TCP (MPTCP) has been standardised by the IETF MPTCP working group in order to provide efficient and fair resource sharing between competing flows. The group has published six RFCs, which discuss different features of MPTCP, such as the architectural components for MPTCP development [78], congestion control mechanism for multipath transport protocols [159], MPTCP implementation instructions [77], MPTCP application interface considerations [169], security considerations for MPTCP [22], and an analysis of the residual threats for MPTCP [23].

MPTCP improves resource utilisation between two multi-addressed endpoints by using multiple paths. It provides the ability to transmit the network traffic over multiple concurrent paths on a single transport-level connection. This provides significant advantages in terms of goodput over traditional TCP. MPTCP works by splitting data from a single TCP connection into multiple subflows and leveraging ECMP [96] to route them over multiple paths. The transport layer in MPTCP is divided into two sublayers, namely, MPTCP and subflow TCP. MPTCP is backwards compatible since it can switch to a conventional TCP connection if the server does not support MPTCP. Besides, MPTCP is implemented to be compatible with existing applications and it provides the same socket interface as TCP [77].

In MPTCP, data can be sent over any of the established subflows over multiple paths. This can cause out-of-order delivery of data packets. This is due to sending packets over different paths with different round-trip times (RTT). MPTCP employs two levels of sequence numbers: subflow sequence number (SSN) and data sequence number (DSN). The SSN is similar to the conventional TCP sequence number, which guarantees that data is transmitted in order over each subflow and it helps in detecting any loss, whereas the DSN is used to reassemble packets received from different subflows. Consequently, MPTCP uses two levels of acknowledgements, connection-level acknowledgements or DATA_ACKs to acknowledge the reception of a packet with expected DSN. In addition, it uses regular TCP acknowledgements to acknowledge the reception of packets sent over the subflow independently of their DSN. Any lost packet can be retransmitted over any of the available subflows [77].

COUPLED has been proposed as an MPTCP congestion control algorithm [159, 194]. COUPLED algorithm works by moving the traffic of a multipath flow to the least-congested path. This algorithm depends on the total congestion window size of all subflows w_{tot}. For each acknowledgement on path r, the congestion window increases by $1/w_{\text{tot}}$, while it decreases by $w_{\text{tot}}/2$ for each loss [159]. In COUPLED, as the least-congested paths are the only paths that are used, this can cause loss of some paths which are underutilised. Therefore, it is important to keep some traffic on these paths (e.g. the congestion window size can be kept 1 packet size as a probe) to re-employ them when the congestion will decrease.

Employing MPTCP in DCNs gives better results in terms of throughput, fairness, and robustness [158, 194]. However, throughput may collapse when multiple subflows collide on the same path. In MPTCP, the completion times of short flows increase as the number of subflows increases. MPTCP can do little to help with the latency of very short transfers. **Maximum MPTCP (MMPTCP)** [109, 106, 107, 108] has been proposed to decrease short flows' completion times, while ensuring high throughput for long flows. MMPTCP initially scatters packets and then switches to the MPTCP after sending a specific amount of data. To prevent spurious fast retransmission, MMPTCP increases the fast retransmission threshold.

2.4.3 pFABRIC: optimal flow completion times

pFABRIC [15] (priority-based packet scheduling and dropping at switches) aims at providing near-optimal flow completion times for short flows by switching packets based on strict priorities. pFabric decouples flow prioritisation from rate control. pFabric's approach is different from DCTCP's [13], where feedback from the network is used to adjust the rate control loop to ensure that high-priority small flows do not see large queues of packets from long flows.

pFabric works very well for short flows as it employs greedy Shortest Remaining Processing Time (SRPT), hence it achieves close to theoretically minimal FCT. SRPT is used as an optimal flow scheduling to assign priorities. pFabric leverages priority queues in switches. In pFabric, end hosts add a number in the packet header that represents its priority. Packets are scheduled and dropped at switches based on their priorities. If the queue overflows, then the switch drops any new incoming packet if it has less priority than the buffered packets. Otherwise, the switch drops the buffered packet with the lowest priority and it buffers the high priority incoming packet. When transmitting, the switch sends the packet with the highest priority. In this way, the switches can operate independently in a greedy and local fashion using priority-based scheduling and dropping mechanism.

pFabric requires specialised network hardware that implements a specific packet scheduling and queue management algorithm. In pFabric, near-optimal FCT can be achieved if each flow can be assigned to a unique priority, so that the link bandwidth can be allocated to flows strictly according to their priorities. There are millions of flows in a DCN. So ideally, we'd like to have the same number of priorities. However, existing commodity switches only provide a very limited number of priority levels. For example, some commodity switches only support eight priorities queues, and in practice, not all of these priority queues can be used for flow scheduling.

In pFabric, long flows suffer a noticeable degradation in the achieved throughput when employing pFabric. In practice, no existing scheme can deliver the near-optimal latency of SRPT at high network load [140]. pFabric approximates SRPT accurately, but it requires too many priority levels to implement with today's switches. Also, pFabric is difficult to deploy as it delegates hosts to correctly prioritise their traffic assuming flow size is known a priori.

2.4.4 pHost

pHost [80] emulates pFabric's behaviour but using only scheduling at the end hosts and hence allows the use of commodity switches. The priority-based packet scheduling happens at end hosts in pHost, while it happens at switches in pFabric. pHost allows end hosts to directly make scheduling decisions, thus avoiding the overheads of Fastpass's [148] centralised scheduler architecture. pHost transport empowers end hosts to perform flow scheduling on the basis of grant assignments.

pHost is a receiver-driven transport that runs very small packet buffers and per-packet ECMP, but does not use packet trimming (as in NDP [66]). pHost uses pull-based packet scheduling using token packets generated from receivers in order to minimize the flow completion time. However, pHost assumes congestion-free network (free congestion core) by using a network with full bisection bandwidth and packet spraying. In addition, pHost does require knowledge of individual flow size in advance.

Destinations can choose which source is entitled to transmit data packets by sending tokens. Sources decide to which destination to reply when receiving multiple tokens. pHost exploits packet-spraying to eliminate congestion by leveraging the property of DCN of full-bisection bandwidth and avoiding explicit path-scheduling

The receivers then assign tokens to the flows, optionally specifying a priority level to be used for the packets in the flow based on performance objectives e.g., using SRPT algorithm for minimizing flow completion time or Earliest Deadline First (EDF) algorithm for deadline-constrained traffic.

2.4.5 PIAS: Practical Information-Agnostic flow Scheduling

Information-aware schemes (e.g, pFabric [15]) require a priori knowledge of flow size or deadline information, while the information-agnostic schemes (e.g., PIAS [24]), make no assumption about the flow information. PIAS leverages multiple priority queues available in existing commodity switches to implement a Multiple Level Feedback Queue (MLFQ), in which a PIAS flow is gradually demoted from higher-priority queues to lower-priority queues based on its priority. Priorities are set on packets of flow at sending end hosts based on the number of bytes it has sent. As a result, short flows are likely to be finished in the first few high-priority queues and thus be prioritised over long flows in general, which enables PIAS to emulate SJF without prior knowledge of flow size information.

The main limitation in PIAS is that the thresholds of the MLFQ are based on using a central server to collect and manage the traffic load information, which is then issued to each end host to determine demoted thresholds for the priority queues. A challenging task associated to MLFQ-based scheduling is the derivation of a set of demotion thresholds that minimises the average and tail FCT. PIAS calculates these thresholds based on traffic information consisting in the CDF of the flow sizes of the workload that will be present in the network. After the derivation of the thresholds, they are distributed and deployed at end hosts. Then, they use these thresholds to perform the packet tagging procedure [24]. However, since traffic in DCN presents time and space variations, a set of thresholds that minimizes the FCT of a given workload might not be adequate for a different one.

2.4.6 Hedera: dynamic flow scheduling in data centres

To address the limitations of ECMP in data centres, Hedera [6] has been proposed as a centralised approach that detects large flows, and then directs them to lightly loaded links. Hedera is based on PortLand [145]. Hedera aims to maximise network utilisation by enhancing PortLand routing and fault-tolerant mechanisms. The flow scheduler collects flow information from switches. This includes the bandwidth utilisation of each flow, and then assigns flows to non-conflicting paths. The scheduler avoids placing multiple flows on a path that cannot fulfil large bandwidth demands [6]. Hedera performs significantly better than the ECMP hash based flow placement [6]. Hedera's limitations are related to the fact that the scheduler needs to detect and mitigate possible congestion in a very short timescale in the high bandwidth and low-latency networks, where traffic patterns are diverse, volatile and unpredictable. Hedera as a centralised solution may face a scalability problem in data centre networks. It has also been shown that Hedera, with a scheduling circle of 500ms, can only achieve similar performance to randomised load-balancing mechanisms [201].

2.4.7 Homa: a receiver-driven low-latency transport protocol

Homa [140] is a receiver-driven low-latency transport protocol using network priorities. Homa provides exceptionally low latency, especially for workloads with a high volume of very short messages, and it also supports large messages and high network utilisation. In Homa, the receiver drives the senders by assigning transmission credits. The credits are used to specify which senders are allowed to transmit to the receiver and to define the network priority to be used during the data transmission. The priority is selected by exploiting information on the flow length distribution so as to prioritise short flows over long ones.

A near optimal policy to minimise the average latency of short packet is the Shortest Remaining Processing Time (SRPT) first scheduler. Such scheduler is hard to implement in practice since the remaining processing time is not available at the server, and it requires too many priority levels. Homa provides a better approximation to SRPT.

2.4.8 QJUMP: Queues don't matter when you can JUMP them!

QJUMP [84], Ethernet Flow Control is a data link layer congestion control mechanism that aims to avoid queuing delays that might affect especially short flows without requiring the size of the flow in advance. QJUMP claims that applications dominated by short flows exhibit low latency variance and that low throughputs require higher priorities whereas applications with high latency variance and high throughput require lower priorities. In order to reduce this network interference, end hosts perform rate limitation in a non-intrusive way; this enables the applications to specify their required priorities. This approach is agnostic in the sense that it does not require to know in advance the size of the flows in order to schedule them. However, QJUMP does not aim at improving the scheduling but reducing the occupation of switch buffers. QJUMP requires an additional API which implies modifications in the applications in order to use it. QJUMP applies QoS-inspired concepts to data centre applications to mitigate network interference. QJUMP requires priorities to be allocated manually on a per-application basis, which is too inflexible to produce optimal latencies.

2.4.9 NDP: Novel Data-centre transport Protocol

NDP [66] takes a different approach to simultaneously achieving both low delay and high throughput. The authors combine a number of existing ideas into a sophisticated clean-slate design. NDP is a receiver-driven transport protocol; receivers can have all required information about the congestion (by receiving data and header packets), therefore they can swiftly react to congestion.

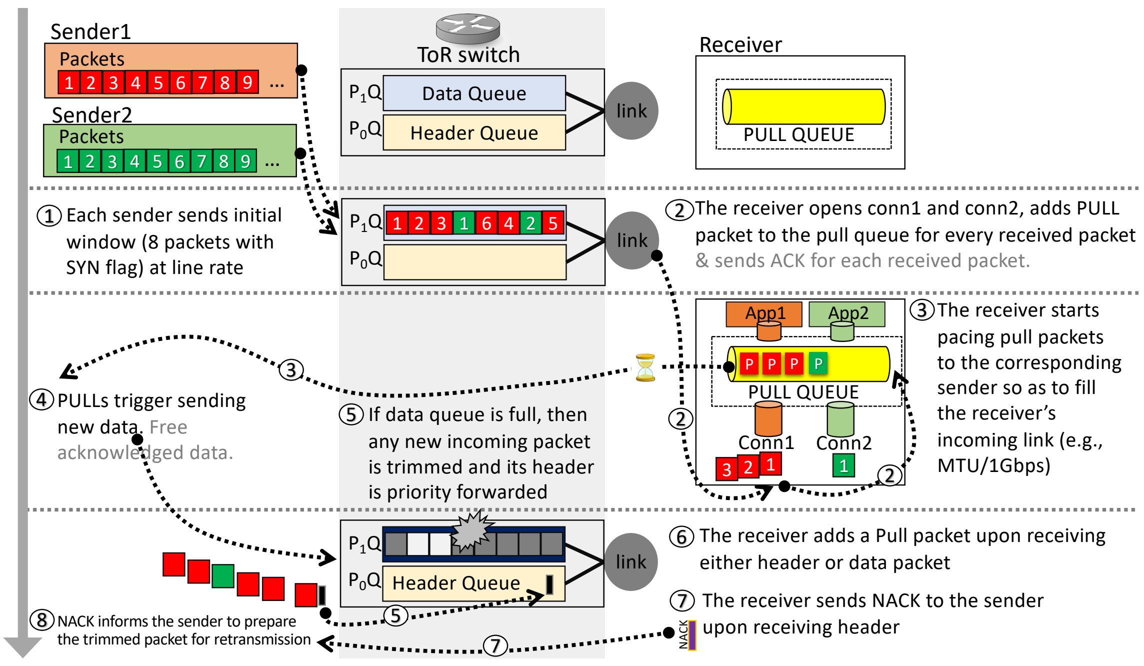

Sender1
Packets
1 2 3 4 5 6 7 8 9 ...
Sender2
Packets
1 2 3 4 5 6 7 8 9 ...
ToR switch
P1Q Data Queue
P0Q Header Queue
link
Receiver
PULL QUEUE
① Each sender sends initial window (8 packets with SYN flag) at line rate
P1Q 1 2 3 1 6 4 2 5
P0Q
link
② The receiver opens conn1 and conn2, adds PULL packet to the pull queue for every received packet & sends ACK for each received packet.
App1 App2
P P P P
PULL QUEUE
Conn1 Conn2
3 2 1
1
③ The receiver starts pacing pull packets to the corresponding sender so as to fill the receiver's incoming link (e.g., MTU/1Gbps)
④ PULLs trigger sending new data. Free acknowledged data.
⑤ If data queue is full, then any new incoming packet is trimmed and its header is priority forwarded
P1Q
P0Q Header Queue
link
⑥ The receiver adds a Pull packet upon receiving either header or data packet
⑦ The receiver sends NACK to the sender upon receiving header
NACK
⑧ NACK informs the sender to prepare the trimmed packet for retransmission

Figure 2.9: NDP protocol

NDP is explained in the numbered steps in Figure 2.9 (see the numbered circles). In *Step 1*, senders are allowed to send their first window of data at line rate without an initial handshake with the receiver as this will minimise the latency cost of the handshake. At the receiver, pull packets are sent back to senders to request new data, or to request retransmissions for missing data (*Step 2*). Because the receiver is in control of pacing the pull packets based on it link's capability, this ensures the aggregate rate seen at the receiver is exactly the available line rate (*Step 3*). This solves the Incast problem. After the initial window sent at line rate, senders can only transmit packets when they see a pull packet and so the network quickly reaches a stable operating point (*Step 4*). Switch queues are kept very short (maximum of 8 packets). If the queue overflows, the packet data is trimmed and the header is priority forwarded (*Step 5*). The receiver adds a pull packet for each received data or header packet, which are then paced from the shared pull queue (*Step 6*). The receiver sends an ACK for each correctly received data packet (this can be a flag in the pull packet). Also, the receiver sends a NACK for each received header (i.e., trimmed packet), and this NACK will inform the sender to prepare the packet that was trimmed for retransmission (*Steps 7&8*).

NDP achieves very good performance, including high throughput and network utilisation with low flow completion time. NDP requires custom switch hardware, which will be deployable when P4 switches [30] are deployed in data centres. However, NDP does not support one-to-many and many-to-one workloads which are very common in today's data centres.

2.4.10 Congestion control mechanisms for RDMA

Unlike TCP, RDMA [98] needs a lossless network; i.e. there must be no packet loss due to buffer overflow at the switches. RDMA over Converged Ethernet (RoCEv2) [98] allows RDMA over an Ethernet network by using PFC (Priority-based Flow Control) [149]. Hosts and switches issue special pause messages when their queues are nearly full, alerting senders to slow down. TIMELY [164] and Datacenter Qau Congestion notification (DCQCN) [203] are congestion control mechanisms for lossless networks. **DCQCN** [203, 155] uses a combination of ECN markings with a QCN-inspired rate-based congestion control algorithm implemented in the NIC [155]. DCQCN is a way of running DCTCP [13] over lossless Ethernet networks. DCQCN reacts to the queue lengths at the intermediate switches to reduce the generation of the PFC pause frames. **TIMELY** [164] relies exclusively on RTT as a congestion metric i.e., it uses delay measurements to detect congestion. TIMELY does not require any switch-support and it is inspired by TCP Vegas [34]. TIMELY takes advantage of modern NIC support for timestamps and fast ACK turnaround to perform congestion control based on precise RTT measurements. Both DCQCN and TIMELY cannot completely prevent pause frames and their negative impact on innocent network traffic. Also, they only focus on low latency, while ignoring network utilization or large-flow performance.

2.4.11 Redundant transmission and coding-based transport protocols

Coding-based transport protocols introduce redundant information by encoding sent packets. This provides faster loss recovery, higher throughput and lower latency. Forward Error Correction (FEC) has been widely studied at the link layer. Network coding extends the concept of encoding a message beyond source coding (for compression i.e., efficiency) and channel coding (for protection against errors and losses i.e., reliability) to implement simple in-network processing with packet combinations in the nodes [182, 100]. Network coding aims at meeting the throughput and latency requirements of applications [71].

End-to-end coding for TCP has been proposed by several studies that suggest redundant transmission to avoid TCP timeout in data centres [50]. Sundararajan et al. [182] suggested placing network coding (NC) in TCP. In [25, 184], the authors explored extending TCP to incorporate FEC in wireless links. Maelstrom [131] is an FEC variant for long-range communication between data centres. Repflow [198] and More-is-less [191] replicate short flows to reduce their delay. Corrective [74] is designed as a TCP extension by employing FEC with coding. Corrective adds aggressiveness to congestion control to do loss recovery before RTO. Corrective can recover a single packet loss in one window without loss detect delay and retransmission delay. Corrective can provide faster loss recovery but it can only deal with one packet loss in one window as its coding redundancy is fixed. Moreover, fountain code is used by FMTCP [49, 48] to improve the performance of MPTCP [157] by recovering the blocked data over multiple subflows. FMTCP uses the sequence-agnostic properties of rateless coding, where coded packets in the same block are sequence-agnostic. Thus, in FMTCP, what matters is receiving sufficient encoded packets rather than which packets are lost/received. LTTP [102] is a UDP-based transport protocol that uses fountain codes (mainly Luby Transform (LT) code [137]) to mitigate Incast in data centre. LTTP employs forward error correction (using LT code) approach instead of retransmissions to tackle the Incast problem when many senders send to the same receiver. LTTP adopts the TCP Friendly Rate Control (TFRC) [90] protocol for congestion control. TFRC is used to adjust the traffic sending rates at servers and it ensures that the sender can send data continuously even if the network is congested rather than stopping sending data for a long time. CAPS [97] solves the out-of-order problem using FEC on short flows to reduce the FCT while it keeps using ECMP on long flows to obtain high throughput. NC-MPTCP [139] introduces packet coding to some but not all subflows. The regular subflows deliver original packets while the coding subflows deliver linear coded packets. The coded packets are used to compensate for the lost and much delayed packets in order to avoid receive buffer blocking.

2.4.12 ICTCP: Incast Congestion Control TCP

ICTCP [195] leverages the idea of improving TCP performance by avoiding packets loss proactively. ICTCP only requires some modifications on the receiver. The receiver is able to perform congestion control based on its prior knowledge of the achieved throughput and the available bandwidth. The implemented algorithm is based on the incoming measured throughput (i.e. the achieved throughput) and the expected throughput. The measured throughput is updated every RTT. The expected throughput of a connection is measured based on the measured throughput, the received window of the connection and the RTT for the same connection. ICTCP adjusts the receiver window (*rwnd*) to avoid TCP Incast depending on the ratio of difference between the measured and expected per connection throughput over expected throughput. When the difference ratio is small, the *rwnd* increases, while the *rwnd* decreases when the difference ratio is large. This algorithm has taken inspiration from TCP Vegas congestion control algorithm [34] but ICTCP is based on the throughput difference ratio, while TCP Vegas uses throughput difference and it modifies *cwnd* at the sender side.

ICTCP performs better than TCP with respect to Incast congestion, as shown in the experimental results in [195]. However, ICTCP does not work efficiently with a large number of senders since it employs per flow congestion control [183]. ICTCP does not exploit the existence of multiple equal-cost paths.

2.4.13 D2TCP: Deadline-aware Datacenter TCP

Flows in DCNs come with real-time constraints on their completion times, typically of the order of tens of milliseconds. One of the issues with DCTCP [13] is that it is a deadline-agnostic; instead, it tries to assign link bandwidth fairly to all flows regardless of their deadlines. As a result, it has been shown [193] that around 7% of flows may miss their deadlines with DCTCP. A number of DCN congestion control have recently proposed to overcome this deficiency by incorporating the flow deadline information into the congestion control algorithm, as in D2TCP [186].

D2TCP [186], like DCTCP, uses ECN which is already effectively supported by existing commodity switches but D2TCP takes into account deadline information before modifying the congestion window size. Similar to DCTCP, each switch marks the CE bit in a packet if its queue size exceeds a threshold K. This information is sent back to the source by the receiver through ACK packets. D2TCP introduces a new variable that is computed on the sender side, called the deadline imminence factor d, which is a function of a flows deadline value. The factor d can be found as: $d = T_c/D$, where T_c is the time needed for a flow to complete transmitting all its data under a deadline agnostic behaviour and D the time remaining until its deadline expires. If $T_c > D$ then the flow should be given higher priority in the network because it has a tight deadline and vice versa. The algorithm behaves differently compared with DCTCP and depending on the value of d, the window size changes as a function of the deadlines. Far-deadline flows gets a smaller window size, so they do not occupy large bandwidth, therefore near-deadline flows get high probability in approaching their deadlines.

D2TCP has been simulated in [186] and the results show that D2TCP reduces the percentage of missed deadlines by 75% when compared to DCTCP [13] and 50% compared to D3TCP [193]. It is also able to coexist with TCP flows without reducing their performance.

In D2TCP, far-deadline flows back-off aggressively, since the priority in bandwidth allocation is provided to near-deadline flows. This approach is inefficient in administrating flows that have data rate requirements, like long flows with deadlines. Initially, long flows backoff for other flows, whereas giving these long flows higher priority in the later stage when their deadlines approach may already not be enough to meet their deadlines requirements [36]. Also, in this approach deadlines are required to be explicitly known.

2.4.14 TCP with Fine Grained RTO (FG-RTO)

In [189], the authors have proposed two solutions for the TCP Incast problem. The first one includes minimum modifications to the regular TCP. They show that the overall throughput in DCNs can be improved significantly by reducing the RTO from its default value (usually 200 ms) to be on the order of microseconds. However, the current operating systems lack the high-resolution timers which are required to support timeouts in the order of microseconds. Although reducing RTO can mitigate the effect of TCP timeouts, it has a side effect of increasing the rate of packet loss. This is expected as reducing the RTO increases the number of competing flows [104].

The second proposed solution is to disable delayed acknowledgements. Delayed acknowledgements are specified in RFC1122 [33]. The idea of this option is to delay the acknowledgement of a data segment from being sent back to the sender. This adds some delay to have data available to be sent back to the sender. In this case the acknowledgement can be piggybacked with a data packet. The delayed acknowledgement timer has a default value of 40ms in Linux and 200ms in Windows [38]. Employing delayed acknowledgement mechanism in DCNs causes significant throughput degradation [189]. However, disabling delayed acknowledgements causes significant increase in the CPU utilisation. In [104], the authors have suggested reducing the delayed acknowledgement to a few milliseconds (around 1ms) instead of completely disabling this option [104].

2.5 Limitations of Existing Approaches

The key takeaway from the description above is that each of the described protocols (discussed in Section 2.4) only focusses on one or some of the limitations discussed in Section 2.3 (as depicted in Figure 2.3). Figure 2.10 shows these protocols and the performance targets that are achieved by each one of them. In particular, DCTCP [13] eliminates Incast and Outcast but it does not improve the network utilisation and it performs less efficiently in ensuring low latency comparing with NDP [66] and HOMA [140]. ICTCP [195], FG-RTO [189] and LTTP [102] are not general-purpose transport protocols, instead they only aim at solving the Incast problem. All of pFabric [15], pHost [80], PIAS [24], QJUMP [84], D2TCP [186] and CAPS [97] protocols show promising improvement in short flows completion times. However, they do not pay attention to long flows throughputs. MPTCP [63] and FMTCP [49, 48] supports multipath transmission but they still suffer from collisions between subflows which might affect the throughput and latency goals. DCQNC [203] and TIMELY [164] are congestion control mechanisms that work for RDMA for reliable fast transmissions, however, they ignore network utilisation and large-flow performance. Among all the discussed protocols, NDP [66] and HOMA [140] are the closest to optimal

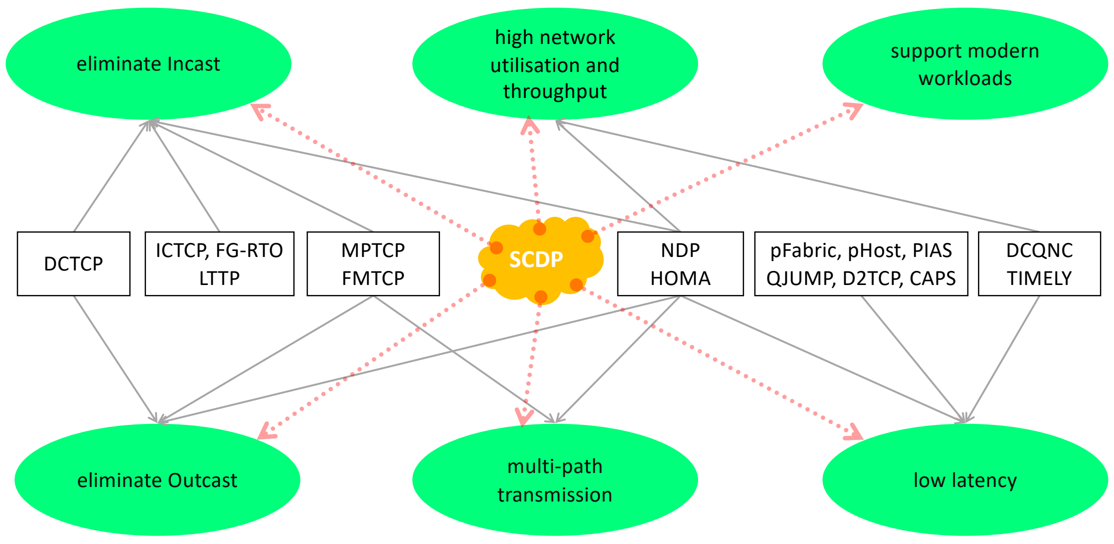


Figure 2.10: Achieved performance goals by each proposed data transport protocols

protocols in tackling all TCP limitations in data centres and achieving all the target performance goals. Their main limitation is that they do not have support for modern data centre workloads.

DCTCP is deployed in an intra-data-centre environment where both endpoints and the switching fabric are under a single administrative domain. However, no real deployment of the other proposed protocols in the data centres has been publicly documented.

In this thesis, we proposed SCDP [10], a general-purpose transport protocol for data centres that is the first to support one-to-many and many-to-one application workloads which are common in modern data centres. SCDP performs at least as well as the state of the art with respect to throughput and flow completion time for long and short unicast flows, respectively. SCDP also eliminates Incast and Outcast, and it improves network utilisation by the benefit of its natural multipath load balancing.

2.6 Data centre network traffic characteristics

Data centres network traffic characteristics are seen as sensitive information by companies, so there is not much work on analysis and characterisation of data centre traffic [105, 26, 167, 27]. Traffic characteristics such as flow size distributions, traffic locality and flow interarrival times are highly correlated with applications [146]. Obviously, understanding intra/inter-data centres traffic characteristics is crucial for effective network management. In this section, we present two main available studies on characterising data centre traffic [26, 167].

Benson et al. [26] present the network traffic characteristics of 10 data centres that belong to three different types of organisations: 5 commercial clouds, 2 private enterprise and 3 university data centres. Each data centre runs a variety of applications including

Web services, custom software applications and intensive Map-Reduce jobs. In this study, the data was collected over several weeks. The main findings in this study are as follows. (1) Flow size: 80% of the flows are smaller than 10 KB in size, and almost all flows are less than 10 MB. (2) Flow live time: 80% of flows are less than 11s long. (3) Interarrival time: 80% of the flows have interarrival times of less than 1ms in private enterprise data centres, while 80% of the flows have interarrival times between 4 ms - 40 ms in the other data centres. (4) Traffic pattern: traffic originating from a rack has an ON/OFF pattern (intervals) with properties that fit heavy-tailed distributions, and traffic that leaves the edge switches is bursty. (5) Active flows: the number of active flows is less than 10,000 per second per rack across all data centres. (6) Traffic locality: 40-90% of the traffic leaves the rack (in the university and private enterprise data centres), while 80% of the traffic coming from servers stays within the rack (in the cloud data centres and due to administrators' applications). (7) Utilisation: link utilisation is higher in the core layer, while the edge layer is lightly utilised. A maximum of 25% of the core links are highly utilised (hot-spots). (8) Losses are not correlated with high link utilisation but are due to temporary bursts.

In [27], the authors studied traffic at the edges of a data centre by examining SNMP traces from routers. The study shows a strong ON-OFF pattern where the packet interarrival follows a log-normal distribution. Also, it shows that utilisation is highest in the core but losses are highest at the edge.

In [167], Facebook reports the characteristics of its traffic. The traffic in this study is one of these applications: Web, Hadoop and Cache applications. The main reported traffic characteristics in this study are as follows. (1) Flow size: median and tail flow sizes for Hadoop, Web Server and Cache applications are reported to be between about 100 KB and 100 MB, 3 KB and 10 MB, 1 KB and 10 KB within racks while 1 KB and 1 MB, 5 KB and 500 KB, 30 KB and 3 MB between racks, respectively. (2) Flow duration: Hadoop flows had a median of about 300 ms and tail of less than 1000 seconds, Web Server flows had a median of about 900 ms and a tail of about 200 seconds, and Cache flows had a median of almost 400 seconds and a tail of almost 800 seconds, respectively. (3) Interarrival time: the median interarrival time of various flow types was between 1ms and 10 ms and the tail was between 10 ms and 100 ms. (4) Traffic locality: the majority of traffic is intra-cluster (57.5%, from caching follower servers), with only 12.9% intra-rack. (5) Active flows: web and cache servers have 100s to 1000s of concurrent connections; Hadoop nodes have 25 concurrent connections on average. (6) authors did not observe an ON/OFF packet arrival pattern at the switches which is suggested to be due to a large number of concurrent destinations, since ON/OFF pattern was observed on a per destination basis.

2.7 Fountain coding

Fountain coding is an information-theoretical approach for efficiently encoding and decoding blocks of data, which can be used to build efficient and reliable data transport protocols. Key properties of fountain coding that we take advantage of in the context of this thesis are as follows.

- **Systematic coding**. In coding theory, codes can be classified into two types systematic and non-systematic codes [173, 174]. A systematic code is any error-correcting code in which the source symbols are among the encoding symbols that are generated. Conversely, in a non-systematic code the output does not contain the source symbols. Linear codes like Hamming, Tornado and Reed-Solomon codes are systematic error-correcting codes [121, 43]. Fountain codes like Luby Transform (LT) [137] codes are non-systematic codes, while fountain codes like R10 (RFC 5053) [123] and RaptorQ (RFC 6330) [67] are systematic codes[112].

- **Rateless erasure coding**. Fountain codes are all rateless codes; i.e. these codes do not exhibit a fixed code rate as they can theoretically produce an infinite number of redundant (or repair) symbols. *In Block Codes (fixed rate)*, k symbols of message are encoded to a pre-assigned number of blocks n i.e., if the code rate is k/n, for every k source symbols, the encoder generates totally n encoding symbols of which $n - k$ are redundant symbols. On the other hand, *Fountain Codes (rateless)* can transform a k source symbols message into an infinite (practically as large as needed) encoded form by generating an arbitrarily large number of redundant symbols [125, 137].

- **Ordering and retransmission are not required**. In fountain coding, the source symbols can be retrieved from any subset of the encoding symbols of size equal to or only slightly larger than the number of source symbols; packet loss and ordering become less important [125, 137]. The name fountain comes from the fact that a fountain code works similar to a water fountain. When filling a bottle from a water fountain, it does not matter which drops fill the bottle, only getting enough drops to fill the bottle matters.

- **Minimal overhead and negligible decoding failure probability**. The overhead is defined as the number of encoding symbols that the decoder needs to collect in order to decode the original data with high probability. If n is the number of encoding symbols, k is the number of source symbols and o is the overhead (the number of redundant symbols), then $n = k + o = (1 + \varepsilon)k$, where $\varepsilon = o/k$. In RaptroQ codes, with two extra encoding symbols $o = 2$, the decoding failure probability is 10^{-6} [67].

- **Low encoding and decoding cost**. For k source symbols, LT codes use on average $O(log(k))$ symbol operations to generate an encoded symbol and $O(klog(k))$ symbol operations to decode the received symbols [173]. Raptor codes can achieve an average of $O(log(1/\varepsilon))$ number of symbol operations per generated encoded symbol and need $O(klog(1/\varepsilon))$ operations to decode the received symbols [173].

- **Fast encoding and decoding**. In [124], the authors report encoding and decoding speeds of over 10 Gbps using a RaptorQ software prototype running on a single core. With hardware offloading RaptorQ codes would be able to support data transport at line speeds in modern data centre deployments.

In a data transport context, fountain coding can be used as follows. A sender uses a fountain encoder to divide the original data into a potentially large stream of encoding symbols. A receiver should be able to recover the original data by collecting enough number of the encoded symbols. Regardless of which symbols were received and which ones were lost. It does not matter which symbols are collected; it only matters that a sufficient number of symbols are received.

Figure 2.11 shows an example of communication using fountain codes. In this example, each sender generates $(1 + \varepsilon)k = 6$ encoded symbols from $k = 4$ source symbols, where $\varepsilon = 0.5$ presents the percentage of redundant symbols that are added to the source symbols. Upon receiving $(1 + \varepsilon)k = 6$ symbols, the receiver has the ability to perform decoding successfully.

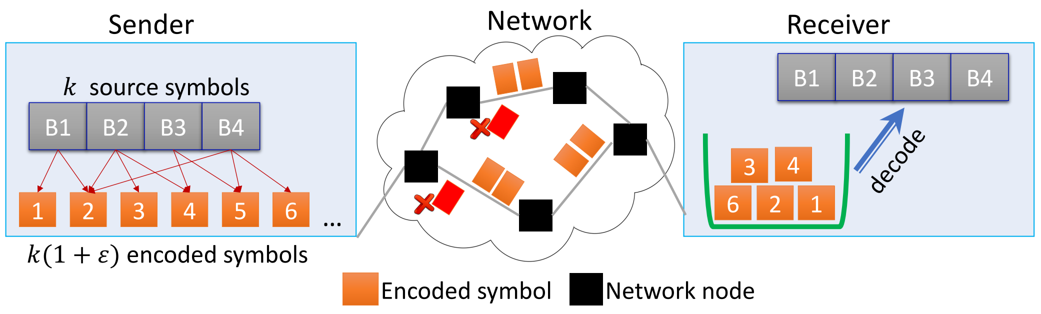


Figure 2.11: Fountain coding-based network [65]

With fountain coding, a number of data transport modes can be supported.

- Unicast transmission, where the sender generates encoded symbols from the source symbols using a fountain encoder. These encoded symbols are placed into packets, which can be transmitted based on a proper flow control mechanism. Whenever the receiver collects enough encoded symbols, it can decode the source symbols.

- One-to-many (multicast) data transmission, where the encoding symbols are transmitted through multicast protocols. The receiver can decode the original data by receiving enough data. Each receiver gets what is enough and stops when they have enough. Again, this requires a well-defined flow and congestion control approach.

- Many-to-one (multisource) data transmission, when a group of senders wants to transmit to a single receiver. The receiver collects enough encoded symbols from the various senders and ideally all the different encoding symbols are equally useful for recovering the original source symbols.

2.7.1 Luby Transform (LT) Codes

LT codes are the first codes in the fountain code family [137]. Below, we discuss briefly both LT encoder and decoder.

LT Encoder

The process of generating an encoding symbol in LT is as follows [137]:

- Choose a degree d that is between 1 and k, where k is the number of source symbols and d is the degree for the encoding symbol and it is selected from an LT degree distribution (Soliton distribution [137]).
- Choose uniformly at random d distinct of the k source symbols.
- Generate the encoding symbol by XORing of d chosen source symbols.

For example, in Figure 2.12, the second encoding symbol E_1 is obtained by XORing the first two input source symbols S_0 and S_1, here d equals 2 which is selected from a LT degree distribution.

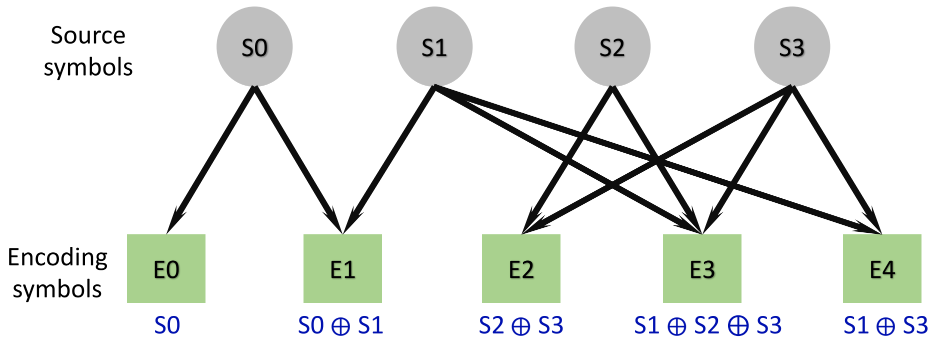


Figure 2.12: LT Encoder

LT Decoder

This decoding algorithm is known as “belief-propagation” [173]. The decoder collects enough encoded symbols that ensures decoding with high probability of success. For the source symbols $S_1, ..., S_k$ and the encoding symbols $E_1, ..., E_n$, the decoder works as follows [173, 137]:

- Collect $k.(1 + \varepsilon)$ encoding symbols.

- Find an encoding symbol with exactly one (degree 1) un-recovered neighbour. This encoding symbol E_i is connected to only one source symbol S_j (if there is no such symbol, then decoding fails).
- Set S_j=E_i
- Add S_j (XOR) to all encoded symbols that are connected to it.
- Remove all the connection lines between the encoded symbols and the source symbol S_j.
- Go to step (2) until all source symbols are determined.

Figure 2.13 shows an example [125] of how LT decoding is processed. In this example, there are three source packets S_1, S_2, and S_3 and for simplicity each packet is just one bit. The receiver receives four encoded packets $E_1E_2E_3E_4 = 1011$. At the first iteration, E_1 is the only encoded symbol of degree 1, so it is used to decode its unique neighbour, by setting $S_1 = 1$ and then removing E_0 (Figure 2.13b). Next, it adjusts the values of the neighbour encoded symbols (E_2, E_4) of the source symbol (S_1) by XORing their values. This results, $S_1 \oplus E_2 = 1 \oplus 0 = 1$ and $S_1 \oplus E_4 = 1 \oplus 1 = 0$, then removes the lines between S_1 and both of E_2 and E_4 (as shown in Figure 2.13c). At the second iteration, E_4 is the only encoded symbol of degree 1, so it is used to decode its unique neighbour, by setting $S_2 = 0$ and then removing E_4 (Figure 2.13d). Next, it adjusts the values of the neighbour encoded symbols (E_2, E_3) of the source symbol (S_1) by XORing their values. This results, $S_2 \oplus y_2 = 0 \oplus 1 = 1$ and $S_2 \oplus E_3 = 0 \oplus 1 = 1$, then it removes the lines between S_2 and both of E_2 and E_3 (as shown in Figure 2.13e). Finally, E_4 is chosen to recover S_3. Hence, $S_3 = 1$ as shown in Figure 2.13f.

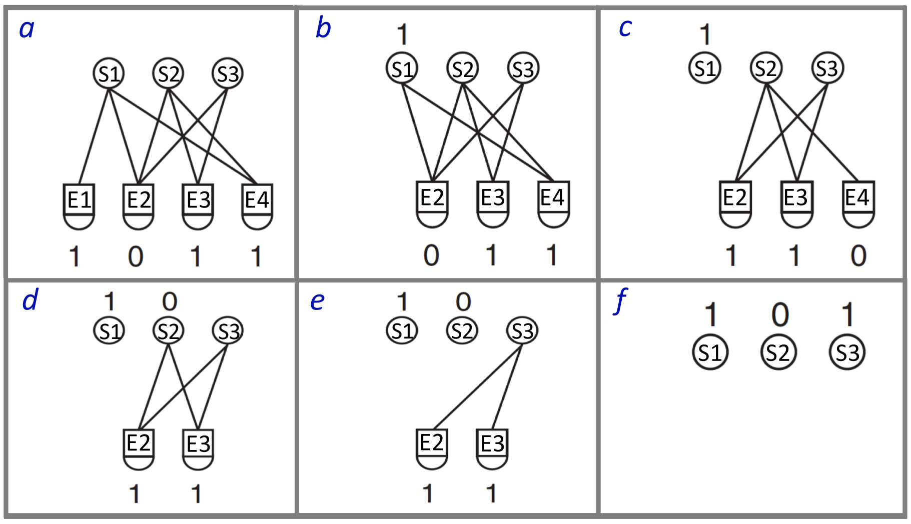


Figure 2.13: LT decoding example [125]

2.7.2 Fountain codes: Raptor Codes (R10 and RaptorQ)

Raptor (Rapid Tornado) codes extend LT codes to improve the encoding and decoding complexity [174]. The encoding of Raptor consists of two phases, as shown in Figure 2.14. Firstly, during the precode phase, k source symbols are encoded into intermediate symbols. The precode can be LDPC (low-density parity-check) code [165]. Secondly, the encoding symbols are generated from the intermediate symbols by using an LT code. The encoding symbols consist of original source symbols and redundant (repair) symbols [174].

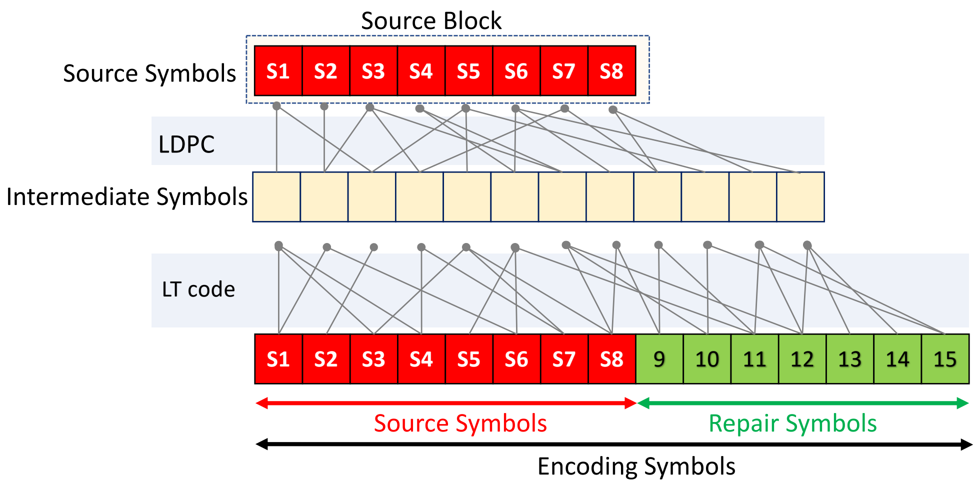


Figure 2.14: Raptor code

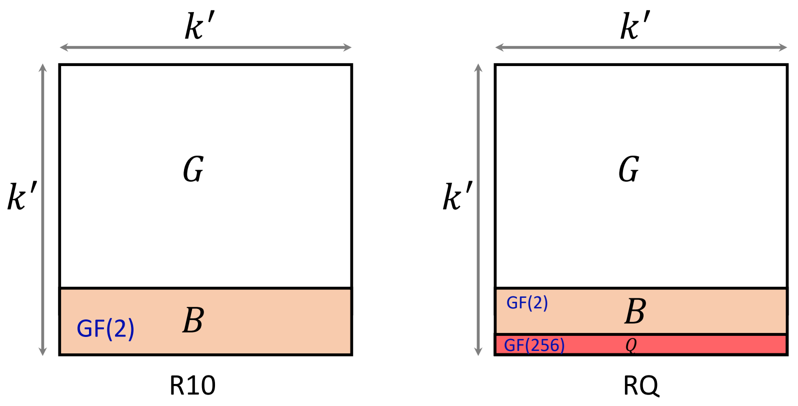


Figure 2.15: The generator matrices of R10 and RQ [173]

Two Raptor codes families have been commercially deployed and standardised, namely R10 [123] and RaptorQ [67] codes. R10 is the first standardised Raptor code that has been adopted into a number of different standards. These include 3rd Generation Partnership (3GPP), Multimedia Broadcast/Multicast Service (MBMS), Internet Engineering Task Force (IETF) and many others [173]. More advanced RaptorQ code [67] is implemented and used by Qualcomm in broadcast/multicast file delivery and fast data streaming applications. R10 and RQ code designs are performed systematically and based on the idea

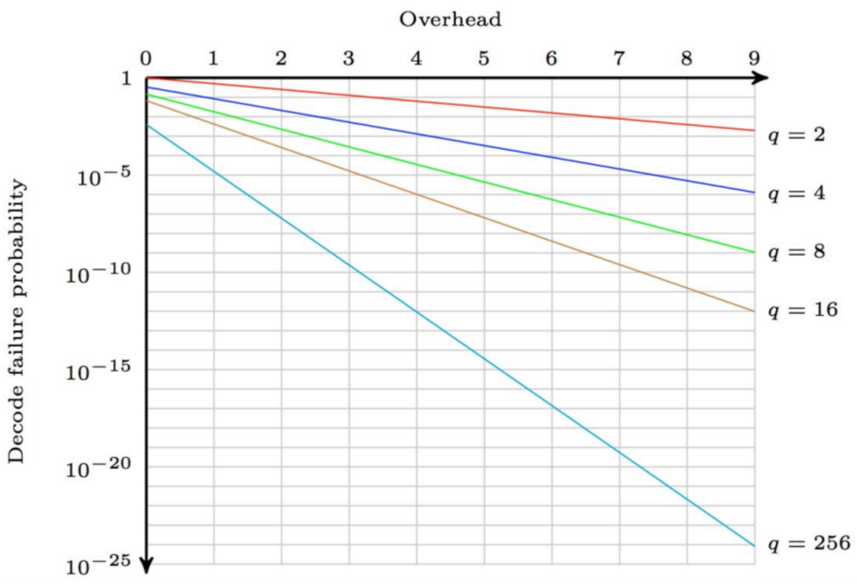


Figure 2.16: RaptorQ Decode failure portability [156]

of inactivation decoding. Thus, their designs are slightly different compared to standard LT codes.

R10 vs RaptorQ. R10 mimics a dense random linear fountain code defined over Galois Field[4] GF(2) whereas RaptorQ uses a mixture of the finite fields GF(2) and GF(256). The majority of the code operates over GF(2) and a tiny minority over uses GF(256). Utilising larger finite fields helps in achieving recovery with lower reception overhead. The generator matrices of these codes have a particular structure, as shown in Figure 2.15. As can be seen, a sparse Graph G is complemented by a denser matrix B (entries from GF(2)) in R10 and additionally by Q (entries from GF(256)) in RaptorQ. The design of B is completely deterministic and consists of two sub-matrices one for sparse LDPC code and one for dense parity check code.

Figure 2.16 shows the decode failure probability curves of different finite fields GF(q) as a function of the overhead. It is obvious that using larger finite fields provide better recovery, but it is more computationally expensive. Conversely, the sum operations in GF(2) is nothing else than the XOR operation.

The two improvements of RaptorQ over R10 codes are the steeper overhead-failure curve and the larger number of supported source symbols per encoded source block. RaptorQ achieves that performance using permanent inactivation decoding and operation over larger field alphabets. RaptorQ offers better coding performance in terms of supporting larger source symbol sizes with less symbol overhead. RaptorQ ensures lower reception overhead as it utilises large finite fields. If the decoder receives two extra encoding symbols, then, on average, the decoder will fail to recover the entire source block at most 1

[4]Galois Field $\mathbf{GF}(p^n)$ is a finite field that contains p^n elements. Representing data as a vector in a Galois Field allows mathematical operations to scramble data easily and effectively. The result of adding or multiplying two elements from the field is always an element in the field.

out of 1,000,000 times [67]. The table in Figure 2.17 compares the properties of both R10 and RaptorQ. RaptorQ can encode up to 56,403 source symbols into a source block in contrast to 8,192 of R10. Also, RaptorQ can generate up to 16,777,216 encoding symbols, 256 times more than R10. For example, if the symbol size is 1024 B, then the maximum source data size that can be encoded by RaptorQ is as follows: $256 \times 56403 \times 1024 \approx 14.78$ GB

Property	R10 (RFC 5053)	RaptorQ (RFC 6330)
Number of source symbols	8,192	56,403
Number of encoding symbols	65,536	16,777,216
Maximum symbol size	65,536	65,536
Number of source blocks	65,536	256
Number of symbols/packet	multiple	1
Recovery ability	Good	Exceptional

Figure 2.17: R10 Vs. RaptorQ

RaptorQ-based data transport example. The encoder generates as many redundant symbols as needed on the fly. Furthermore, since the code is systematic, the encoding symbols include both the source symbols and the repair symbols. Therefore, source symbols can be placed directly in the transmitted packets. The redundant symbols are sent to the network as they are generated. The benefit is that in the case where there are no packets lost, there is no need for decoding and the data can be directly delivered to the application. Figure 2.18 shows an example of communication using RaptorQ. The data is first partitioned into source blocks; the actual way of doing this is left to the application. These blocks are processed independently by the encoder, where they are divided into K equal sized units called source symbols. The decoder starts decoding to recover the source block after receiving any subset of encoding symbols whose size is equal or slightly larger than K. The received repair symbols are used to compensate for any lost symbols. These extra repair symbols are the overhead symbols. The code overhead is defined as the minimum number of repair symbols required to start the decoding process. The code overhead can be agreed upon between the sender and receiver, the minimum value is $o = 0$. RaptorQ can successfully decode with a high probability with overheads as low as 0 to 2 (see Figure 2.16) [173].

RaptorQ data partitioning. The required input parameters to perform encoding and decoding are the total size of the input object, the symbol size and the number of source blocks. Moreover, the RFC also explains an example partitioning algorithm (Section 4.3 in [67]). The algorithm makes the symbol size equal to the maximum payload of a packet (i.e. the MTU minus the headers), ensuring that a recommendation is followed where each packet contains exactly one symbol (see Figure 2.19). It then distributes the data through a number of source blocks, trying to maximise their size, while taking into

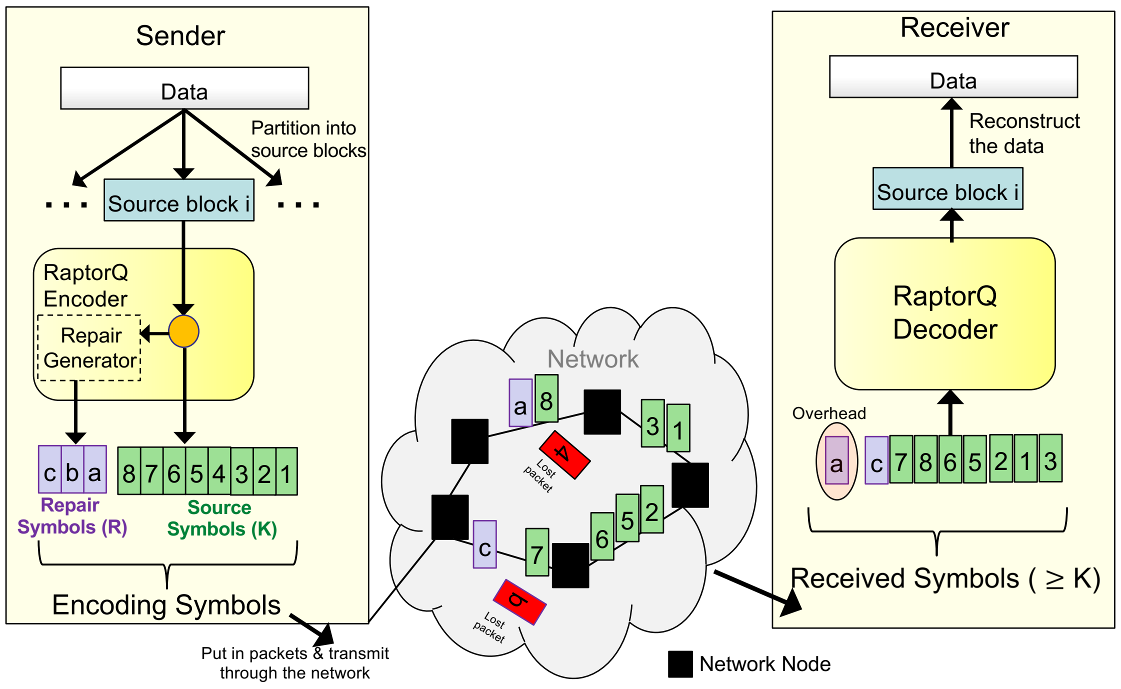


Figure 2.18: RaptorQ-based data transport

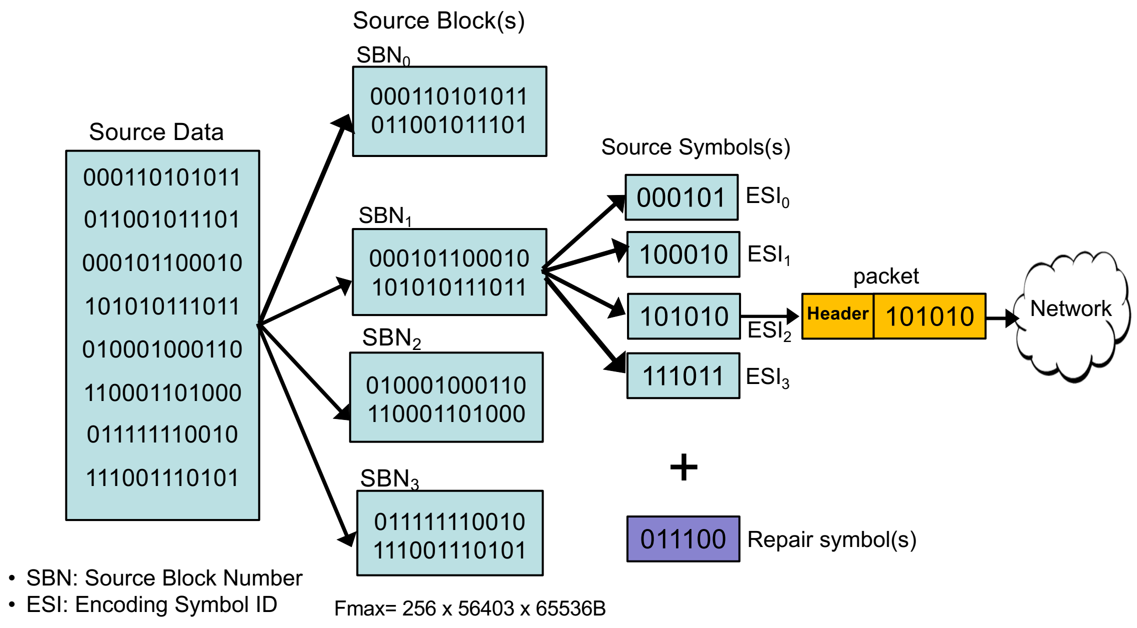


Figure 2.19: RaptorQ data partitioning

consideration the memory limitations of the receivers. After that, the source block is further divided into K source symbols (see Figure 2.19).

The encoding process in RaptorQ codes. The encoding process starts by constructing the extended symbols; then, a precode matrix A (the generator matrix) is generated based on the input parameters; the intermediate symbols are obtained by solving a system of linear equations; a 'Tuple Generator' is used to produce the repair symbols.

Finally, the encoding symbols are formed by combining the source symbols together with the repair symbols. Finding the inverse of matrix A is the most time consuming process in the whole RaptorQ encoding and decoding processes. Hence, an effective and optimised mechanism is required. RaptorQ uses an optimised permanent inactivation technique.

The decoding process in RaptorQ codes. The decoding process is very similar to encoding. The decoder needs to know the structure of the source block; therefore, this configuration information must be conveyed by the transmitter. If all source symbols arrive, decoding is not required. Otherwise, the construction of a system of linear equations takes place. If additional repair symbols are received, they may also take part in the system of equations, ensuring a much greater probability of successful decoding (these are the overhead symbols). The specification of the RaptorQ code proposes an efficient decoding algorithm. It is built around the idea of inactivation decoding. This mechanism combines the low-complexity of belief-propagation (BP) decoding with the decoding guarantee of Gaussian elimination (GE) decoding.

2.8 Network Simulator

OMNeT++ [188] is an excellent simulation environment for developing models for data centre networks and respective protocols. This is possible through the INET Framework, which is built on top of the simulation core provided by OMNeT++. OMNeT++ and INET is built around the concept of modules that communicate by message passing. Protocols are represented by components, which can be combined to form hosts, routers, switches and other networking devices. What makes this framework ideal for evaluating DCN protocols is that new modules can be easily integrated with the existing modules. DCN topologies (e.g. FatTree [5]) can be easily built and parametrised using OMNeT++ NED language.

Recently, some DCN-related research has been based on OMNeT++/INET [140, 12, 55, 9]. Large-scale simulations are crucial for the DCN research community given that access to real-world deployments is very difficult. Developing models for DCNs in OMNeT++ would ensure reproducibility, revisability (dynamic debugging and profiling) and control over the studied traffic workloads (generating realistic traffic workloads in a deterministic fashion) [166].

2.9 Internet Traffic Characterisation and Modelling

Recently, there has been an increasing demand on high performance services in the Internet; these services include data, voice and video transmission. In the context of IP networks, the validation of network performance requirements depends on the examination of QoS metrics such as delay, jitter, packet loss, availability and throughput [99]. These metrics are described in a committed contract between the users and the service

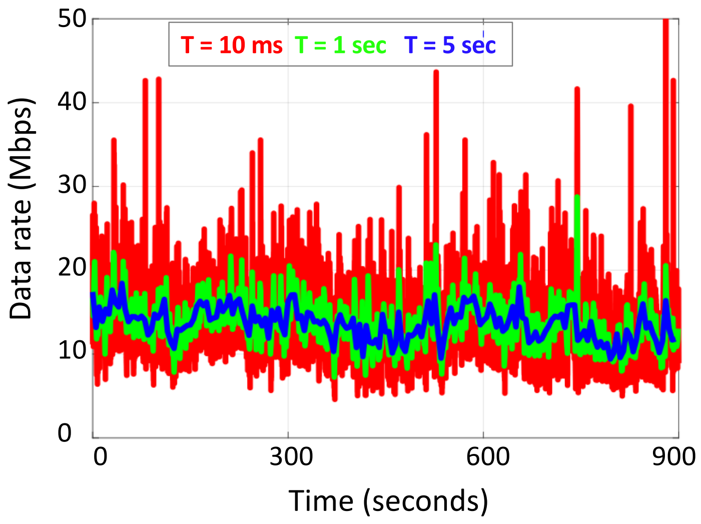


Figure 2.20: The data rate of an Internet traffic trace at different timescales

providers which is known as service level agreement (SLA). This indicates the importance of allocating sufficient bandwidth in the network. In general, traffic characterisation is important for network design, planning, deployment and management; e.g. for traffic billing and network dimensioning.

2.9.1 Traffic fluctuations

It has been shown that traffic volume fluctuates significantly at small aggregation times [127, 3, 134, 11]. The timescale of traffic aggregation is therefore critical in assessing the volume of the underlying traffic and provisioning the relevant network links. Figure 2.20 shows the data rate bps of a Mawi trace [130] over an interval of 900 sec at different timescales: 10 ms, 1 sec and 5 sec. It is obvious that more fluctuations appear at small aggregation times. The more the fluctuations, the more the data rate values are far from the mean, which indicates more variation. Therefore, the conventional techniques of bandwidth allocation are imprecise at small timescales and this could result in provisioning that would break agreed SLAs.

The importance of considering these fluctuations is to ensure that any modelling process will contain all the properties of the traffic. Thereby, the quality of service (QoS) of the network will not be affected by the mismatching between the real traffic and the reference model. Consequently, the seeking of an accurate Internet traffic model is one of the main challenges for network planning.

2.9.2 Representing traffic volumes using the Gaussian model

Historically, network traffic has been widely assumed to follow a Gaussian distribution. In [134, 52, 53], the authors studied network traces and verified that the Gaussianity assumption was valid (according to simple goodness-of-fit tests they used) at two different timescales. In [81], the authors studied traffic traces during busy hours over a relatively long period of time and also found that the Gaussian distribution is a good fit for the captured traffic. Schmidt et al. [51] found that the degree of Gaussianity is affected by short and intensive activities of single network hosts that create sudden traffic bursts. All the above mentioned works agreed on the Gaussian or 'fairly Gaussian' traffic at different levels of aggregations in terms of timescale and number of users.

2.9.3 The failure of the Gaussian model in modelling traffic volumes

There has been some researches that demonstrates the failure of the Gaussian model in representing the Internet traffic volumes [3, 110, 103, 60, 163]. This failure comes from the fact that Internet traffic is bursty on a wide range of time scales. For example, the authors in [110, 60] examined the levels of aggregation required to observe Gaussianity in the modelled traffic, and concluded that this can be disturbed by traffic bursts. The work in [3, 199] reinforces the argument above, by showing the existence of large traffic spikes at short timescales which result in high values in the tail. The fact that Gaussian distribution characterises several aggregated traffics is based on the central limit theorem [51]. However, this theory is valid for independent and identically distributed (iid) random processes and it fails if there are dependencies between any combinations in the distribution.

2.9.4 Heavy-tailed traffic

Deciding whether Internet flows could be heavy-tailed became important as this implies significant departures from Gaussianity. The authors in [76] provided robust evidence for the presence of various kinds of scaling, and in particular, heavy-tailed sources and long-range dependence in a large dataset of traffic spanning a duration of 14 years. Also, there is a large body of work [115, 168, 199, 93] which shows that Internet traffic is heavy tailed. These studies [60, 120, 143, 126] show that Internet traffic is characterised by long-tailed distributions such as Log-normal, Pareto, Weibull, Generalized Extreme Value (GEV) and log-gamma distributions.

2.9.5 Link dimensioning: bandwidth over-provisioning and provisioning

The designing of an accurate Internet traffic model is crucial as this model plays a critical role in network planning. Bandwidth provisioning is a commonly used bandwidth allocation mechanism. The main idea in bandwidth provisioning is to allocate sufficient bandwidth to the link until achieving acceptable performance, which ensures that the SLA requirements are met[153, 53]. In the conventional methods of bandwidth provisioning, operators just apply rules of thumb, such as **bandwidth over-provisioning** by upgrading the link bandwidth to 30% of the average traffic value [153]. This ensures that no traffic congestion will occur in the link. The drawback of this mechanism is that it can provide the link more bandwidth than is actually needed, unnecessarily increasing the deployment cost. On the other hand, the **bandwidth provisioning** approach provides the link by the essential bandwidth that guarantees the required performance [187]. Deploying the bandwidth provisioning approach requires an accurate model of the traffic. Running bandwidth provisioning approach over a Gaussian model does not necessarily achieve the target performance, and attention has to be paid to the tail values.

Meent et al. [153] proposed a new bandwidth provisioning formula, which relies on the statistical parameters of the captured traffic and a performance parameter. This formula calculates the minimum bandwidth that guarantees the required performance, according to an underlying SLA. They assumed that Internet traffic can be characterised by a Gaussian distribution. Their assumption about the applicability of a Gaussian distribution to represent the Internet traffic is based on related work such as [134, 79]. However, this work has not investigated the validation of the Gaussianity assumption. Hence, this formula may provide inaccurate bandwidth allocation for links with heavy tailed traffic [8]. Designing a favourable bandwidth provisioning scheme requires a better model that can fit the traffic at different aggregation times.

2.9.6 Network traffic billing: the 95th percentile

The 95th percentile method is used widely for network traffic billing. Dimitropoulos et al. [58] found that the computed 95th percentile is significantly affected by traffic aggregation parameters. However, in their approach they do not assume any underlying model of the traffic; instead, they base their study on specific captured traces. Stanojevic et al. [180] proposed the use of Shapley value for computing the contribution of each flow to the 95th percentile price of interconnecting links. In [82, 113, 35, 197] authors propose calculating the 95th percentile using experimental approaches. Xu et al. [196] assume that network traffic follows a Gaussian distribution“through reasonable aggregation” and propose a cost-efficient data centre selection approach based on the 95th percentile.

2.9.7 Modern statistical framework for fitting Internet Traffic

Employing a robust statistical approach in finding the best model for Internet traffic volume is crucial. As discussed above, the importance of this model can be shown through two sample traffic engineering problems: firstly, predicting the proportion of time a link will exceed a given capacity. This could be useful for provisioning links, secondly, predicting the 95th percentile transit bill that ISP might be given. The Internet networks traffic (packet-based networks) shows clear fluctuations over a wide range of timescales (as discussed above). The conventional and simple approaches for testing distribution fitting (e.g., quantile-quantile plots) gives inaccurate results (as discussed in Appendix A in [44]).

For a random variable x of a power-law distribution, the probability distribution can be written as follows, $p(x) \sim x^{-\alpha}$, where α is the scaling parameter. The linear form can be derived by applying the logarithm as follows,

$$log(p(x)) = \alpha log(x) + c \tag{1}$$

The value of α is obtained from the log-log plot, where α is the slope of the fitted line (using least-squares method) in this plot. This approach has several drawbacks, such as requiring large sample size and being sensitive to fluctuations in the tail of the distribution [119][44].

Clauset et. al. approach: introduction. In [44], the authors present a well-defined statistical framework for testing power-law behaviour in empirical data[5],[6]. We use this approach to find out which distribution is the best fit for Internet traffic volumes. The approach provides strong evidence whether power-law (Pareto) or another alternative distribution (e.g., log-normal, exponential, Weibull, ..) is favoured in fitting the traffic. As shown in Figure 2.21, this approach is based on introducing a normalised constant (scaling factor that ensures that the total probability is 1) C to the probability density function of each distribution. The new definition of the probability destiny function ensures better representation for the large fluctuations that occur in the tail of the distribution. The constant C can be derived in term of x_{min} (see Figure 2.21). Hence, the power-law distribution can be defined as follows: $p(x) = Cx^{-\alpha}$, i.e.,

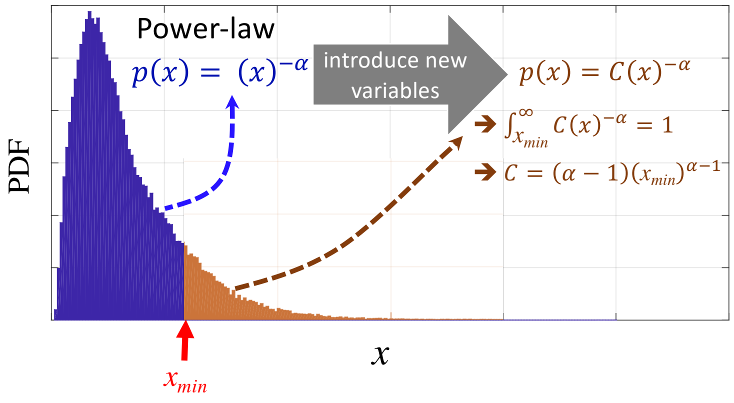

Power-law
$p(x) = (x)^{-\alpha}$
introduce new variables
$p(x) = C(x)^{-\alpha}$
$\rightarrow \int_{x_{min}}^{\infty} C(x)^{-\alpha} = 1$
$\rightarrow C = (\alpha - 1)(x_{min})^{\alpha-1}$
PDF
x
x_{min}

Figure 2.21: The basic idea in Clauset et. al. approach

$$p(x) = \frac{\alpha - 1}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-\alpha} \tag{2}$$

where x_{min} is the estimate of the lower bound of the power-law behaviour and α is the scaling exponent. The power-law distribution applies to the elements above x_{min}, i.e, $x \geq x_{min}$. Similarly, all other distributions (e.g., log-normal) can be redefined in that way (see Table 2.1 in [44]).

Clauset et. al. approach: steps. Figure 2.22 describes all the steps in this

[5] `www.tuvalu.santafe.edu/~aaronc/powerlaws/`
[6] `www.pypi.org/project/powerlaw/`

approach, which we briefly discuss in this section. In this approach, the power-law distribution has the probability density function as defined in Equation 2.

Step 1(a): Estimating the scaling parameter α. Finding the value of α that makes the power-law model most likely to have generated our data. The estimated value of α is found by using Maximum Likelihood Estimation (MLE), where the likelihood function is

$$L(\alpha) = \prod_{i=1}^{n} p(x) \tag{3}$$

where n is the number of observations. Thus, the estimated $\hat{\alpha}$ is the value α that maximises the log-likelihood function, which can be found by solving this first derivative equation: $\frac{dlog(L(\alpha))}{d(\alpha)} = 0$, which gives

$$\hat{\alpha} = 1 + n \left[\sum_{i=1}^{n} ln \left(\frac{x_i}{x_{min}} \right) \right]^{-1} \tag{4}$$

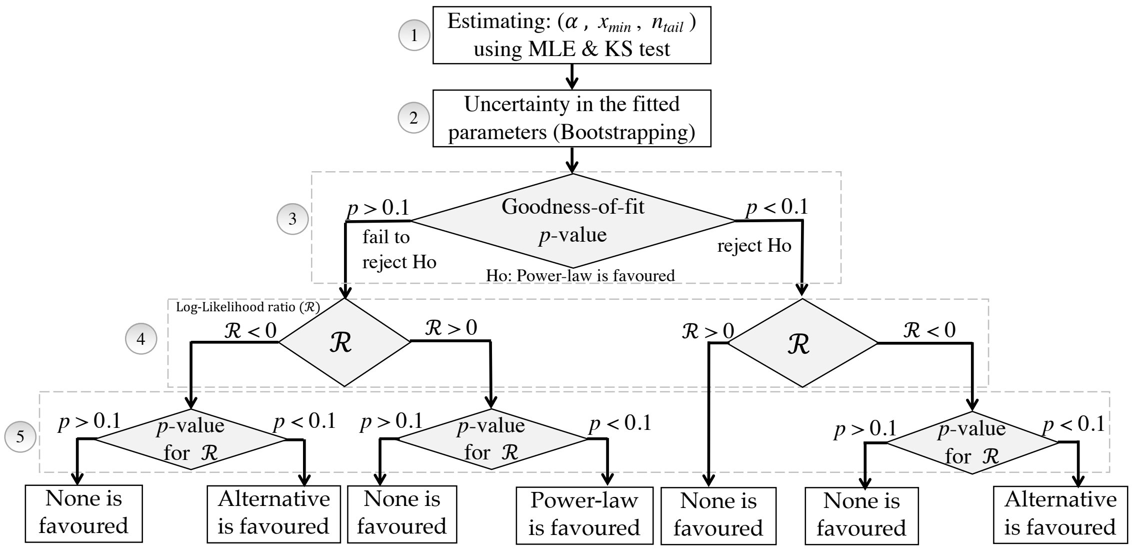


Figure 2.22: Finding the best fitted distribution based on the power-law approach

Step 1(b): Estimating x_{min} value. The estimated value of x_{min} is calculated using the Kolmogorov-Smirnov(KS) D-statistic test [128]. The value of D represents the distance between the CDF of the input data $S(x)$ and the CDF of the power-law distribution $p(x)$. Now, the estimated $\hat{x}_{min}$ is the value of x_{min} that minimises the distance D, as in Equation 5.

$$D = \max_{x \geq x_{min}} |S(x) - P(x)| \tag{5}$$

Estimating parameters in this approach is more rigorous than the conventional linear fit base on log-log plots.

Example. Here, we present an example of how the power-law test is used to determine which distribution is the best to fit captured Internet traffic. The tested trace is a 15-minute long Mawi trace aggregated at a timescale of 100 ms. The trace contains 9000 data rate samples. The data rate PDF of this trace is shown in Figure 2.23. The estimated parameters of applying the power-law test on this trace are as follows: $\hat{\alpha} = 5.6867$, $\hat{x}_{min} = 202052640$ bps and the number of samples above x_{min} is $\hat{n}_{tail} = 1746$. The uncertainty in the estimated parameters is quantified using resampling (bootstrapping) [61], which when applied in the above example, gives these uncertainty values: $\hat{\alpha} = 5.6867 \pm 0.4465$, $\hat{x}_{min} = 202052640 \pm 29367788$ and $\hat{n}_{tail} = 1746 \pm 791$.

Step 2: Uncertainty in the estimated parameters α and x_{min}. This is done by using resampling methods such as bootstrapping.

Step 3: Goodness-of-fit test. The goodness of fit of the power-law distribution has to be evaluated before concluding that a power-law is a good description of the traffic. The null hypothesis **Ho** here is that the sample is drawn from a power-law distribution, while the alternative hypothesis **H1** describes the case that the sample is not drawn from a power-law distribution.

Steps 4&5: Alternative Distributions. The likelihood-ratio test is used for comparing the goodness of fit of two different distributions, which are the power-law distribution and an alternative distribution (e.g., log-normal, exponential, Weibull, ..). Hence, this ratio is defined as the ratio of the likelihood function of each model, which is given by

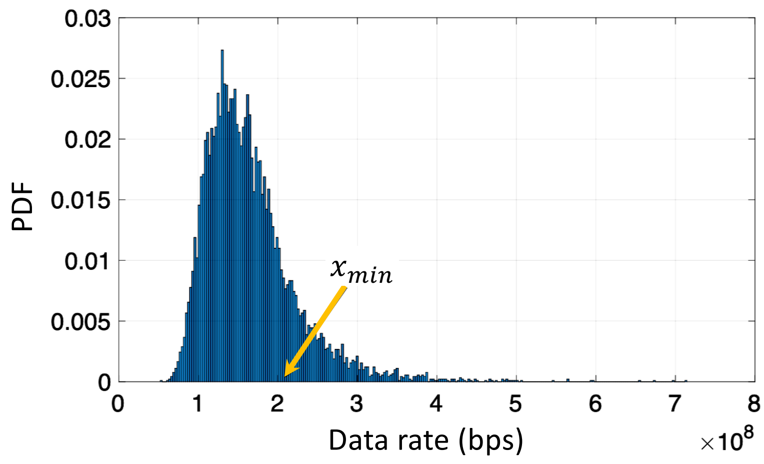


Figure 2.23: Data rate PDF of a Mawi trace

$$R = \frac{L_1}{L_2} = \frac{\prod_{i=1}^{n} p_1(x)}{\prod_{i=1}^{n} p_2(x)} \tag{6}$$

where $p_1(x)$ is the power-law likelihood function and $p_2(x)$ is the alternative distribution likelihood function. The log-Likelihood ratio is obtained as

$$\Re = \sum_{i=1}^{n} [log(p_1(x_i)) - log(p_2(x_i))] \tag{7}$$

If $\Re > 0$, then the power-law distribution is favoured, while if $\Re < 0$, then the alternative distribution is favoured. Equation (7) gives the value $|\Re|$, which is the measured log-likelihood ratio as magnitude. In order to support one hypothesis (either power-law or alternative), then the magnitude $|\Re|$ has to be large enough (not close to zero). This can be tested by finding the p-value. By the Central Limit Theorem, the sum $\Re$ becomes normally distributed as n gets very large, i.e., $\Re \sim N(n\mu, n\sigma^2)$. Thus, the p-value is defined as the probability that $|\Re|$ is not close to zero:

$$p = \int_{-\infty}^{|\Re|} N(n\mu, n\sigma^2) dR + \int_{|\Re|}^{\infty} N(n\mu, n\sigma^2) dR \tag{8}$$

A small p-value ($p < 0.1$) means that the value of $\Re$ can be trusted to make a conclusion about which distribution is a better fit to the data. In contrast, a large p-value ($p > 0.1$) indicates that $|\Re|$ is close to zero and there is nothing to be concluded from the likelihood ratio test.

The test returns the value of the normalised log-likelihood $\Re$ ratio and the p-value. If $\Re < 0$, then the alternative distribution is favoured, while if $\Re > 0$, then none is favoured. A small p-value ($p < 0.1$) means that the value of $\Re$ can be trusted to make a conclusion about which distribution is a better fit to the data. In contrast, a large p-value ($p > 0.1$) indicates that $|\Re|$ is close to zero and there is nothing to be concluded from the likelihood ratio test.

2.9.8 The studied traces

Understanding the characteristics of Internet traffic by investigating real Internet traces from different networks is a right step towards designing and developing new protocols. The used dataset in this work includes various Internet traces from wide-area traffic (see Chapter 5). Our dataset does not include any data centre or mobile/wireless access networks traces. However, the techniques that we suggest to study Internet traffic could be used for any network traffic regardless of the network type.

Chapter 3

SCDP: Systematic Rateless Coding for Efficient Data Transport in Data Centres

Preface: paper 1&2

This chapter includes our paper on developing SCDP [10][1], a fountain coding-based data transport protocol. Through large-scale simulations, we demonstrate the superiority of SCDP in comparison to TCP and NDP for a large and diverse set of traffic workloads and networking scenarios. For completeness, we also provide our early publication that set the foundation for researching SCDP [9].

- Mohammed Alasmar, George Parisis and Jon Crowcroft, "Polyraptor: embracing path and data redundancy in data centres for efficient data transport". In Proceedings of ACM SIGCOMM 2018 (Poster Sessions), Budapest, Hungary [9].
- Mohammed Alasmar, George Parisis and Jon Crowcroft, "SCDP: Systematic Rateless Coding for Efficient Data Transport in Data Centres". IEEE/ACM Transactions on Networking 2019 (submitted) [10].

Contributions from Co-Authors

The research presented in the papers was driven by me. My co-authors contributed much in shaping the presented arguments, design solutions and experimental evaluation. They have also provided constructive feedback about the manuscript and shepherded me through the submission process. The initial abstract idea of employing fountain codes for data transport in data centres was published by my supervisor in a HotNets paper (2013) [65]. Both papers in this chapter are inspired by that position paper, and substantially built on in to provide a full realisation of a real-world data transport protocol for data

[1]`https://arxiv.org/abs/1909.08928`

centres. My supervisor provided suggestions and feedback on the developed approaches and experiments in these papers. He also helped in forming the motivation of this work by directing me to the literature and related work on this topic. In addition, he guided me through the design and implementation phases of this research.

Polyraptor: Embracing Path and Data Redundancy in Data Centres for Efficient Data Transport

Mohammed Alasmar
Department of Informatics
University of Sussex
M.Alasmar@sussex.ac.uk

George Parisis
Department of Informatics
University of Sussex
G.Parisis@sussex.ac.uk

Jon Crowcroft
Computer Laboratory
University of Cambridge
Jon.Crowcroft@cl.cam.ac.uk

ABSTRACT

In this paper, we introduce Polyraptor, a novel data transport protocol that uses RaptorQ (RQ) codes and is tailored for one-to-many and many-to-one data transfer patterns, which are extremely common in modern data centres. Polyraptor builds on previous work on fountain coding-based transport and provides excellent performance, by exploiting native support for multicasting in data centres and data resilience provided by data replication.

CCS CONCEPTS

• **Networks → Data center networks**; **Transport protocols**; *Network performance analysis*; *Network simulations*;

KEYWORDS

Datacenter Storage; Data Transport; Fountain coding

ACM Reference Format:
Mohammed Alasmar, George Parisis, and Jon Crowcroft. 2018. Polyraptor: Embracing Path and Data Redundancy in Data Centres for Efficient Data Transport. In *SIGCOMM Posters and Demos '18: ACM SIGCOMM 2018 Conference Posters and Demos, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234200.3234222

1 INTRODUCTION

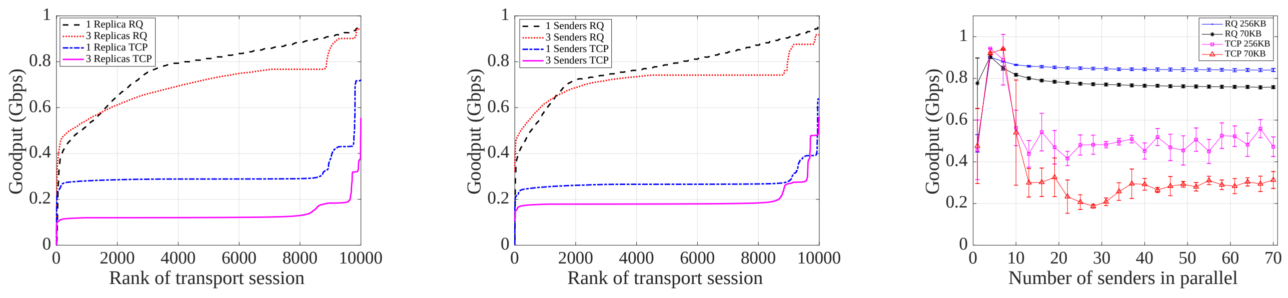
Data centres support the provision of core Internet services, such as search, social networking, cloud computing and video streaming. Data Centre Networks (DCNs) consist of a large number of commodity servers and switches, support multiple paths among servers and very large aggregate bandwidth. TCP is ill-suited for meeting the throughput and latency requirements of applications in DCNs. Exploiting multiple paths and maximising resources' utilisation, while network congestion is fairly dealt with, has been a prominent research area [1][6][7][4][8]. A range of application workloads in modern data centres involve one-to-many and many-to-one traffic exchange. For example, distributed storage systems, such as GFS [9], replicate data blocks, but clients are constrained by the underlying unicast transport protocol when storing data to multiple servers (one-to-many) and fetching data that is available on multiple servers (many-to-one). Partition-aggregate application workloads are similarly constrained, as they make use of underlying distributed storage systems. Polyraptor builds on our previous work [3] and is tailored for one-to-many and many-to-one data transfer patterns, supports multi-path transport, eliminates Incast and can work well with shallow buffers in network switches (section 2). Polyraptor uses RQ codes [5] and follows a receiver-driven approach for flow and congestion control, which is reminiscent to NDP [6]. We have implemented a simulation model of Polyraptor [1] and compared its performance to standard unicast data transport (section 3).

2 DESIGN

Polyraptor employs a receiver-driven communication model, where receivers actively manage the rate at which encoding symbols arrive by explicitly requesting symbols from senders. RQ codes are rateless and systematic; encoding symbols consist of the source symbols (i.e. original data fragments), along with a potentially very large number of repair symbols. In Polyraptor, source symbols are sent at the beginning of a session, followed by repair symbols, as required by receivers. In the absence of loss, source symbols are immediately passed to the application without inducing any penalty in terms of decoding latency; this is particularly desirable for short flows that are commonly latency-sensitive. RQ codes have excellent performance in terms of network overhead, decoding latency and failure probability[5][2].

Polyraptor sessions. A Polyraptor session may involve one sender and multiple receivers or multiple senders and one

[1]Our Polyraptor OMNet++ model: https://github.com/mzsala/polyraptor.
[2]Decoding a source block fails only 1 in 1,000,000 when the receiver collects *n + 2* encoding symbols, *n* being the number of original fragments [5].



(a) Multicast **(b) Multi-source** **(c) Incast**

Figure 1: Goodput results at a 250 servers FatTree topology (1GB link speed & 10μs link delay). 20% of the sessions are background traffic. The presented results are for the rest 80% of the sessions (4 MB each) with arrival times follow a Poisson process with $\lambda = 2560$. Session (flow) scheduling follows a permutation traffic matrix. Error bars in (c) represent 95% confidence interval. Each experiment is the average of 5 repetitions using different seeds.

receiver (unicast data transport is a specialisation of one of the above scenarios). A sender first sends a whole window of encoded symbols at line rate for the first RTT; receivers then take control of the data transfer by requesting encoded symbols (by sending pull requests). We adopt NDP's switching architecture [6], which supports two different packet queues: a priority header queue and a data queue. When the data queue overflows, incoming encoding symbols are trimmed and the resulting headers get priority forwarding. The data transport layer at each receiver has only one pull queue shared by all sessions. A pull request is added to this queue upon receiving a full or trimmed symbol. The receiver then paces pull packets across all sessions, so that the aggregate data rate matches the receiver's link capacity. These pull packets trigger the sending of new encoded symbols. A lost symbol does not have to be re-requested. Instead, a new symbol will contribute to the decoding process equally to the lost one. This, along with symbol trimming, is crucial for supporting an Incast-free protocol. Packet loss and out-of-order packets don't hurt performance as they do in TCP, thus there is no need to extensively buffer packets to minimise losses; symbols can be sprayed in the network, exploiting all available (equal-cost) paths.

Multi-source transport. In Polyraptor, a receiver can pull encoding symbols from multiple senders. RQ symbols are equally useful for decoding the original data, if no duplicate symbols are received. This can be achieved without any coordination; senders independently seed the underlying pseudorandom generator that is used to encode symbols [5], therefore collectively producing statistically unique symbols. Senders initially select and send a subset of source symbols (exploiting the systematic nature of RQ codes), before sending statistically unique repair symbols. The number of senders is known at session establishment, therefore a simple partitioning of the source symbols would ensure absence of duplicate symbols at the receiver side. Multi-source transport enables a natural load balancing mechanism where each server contributes symbols at its available capacity.

Multicast transport. The rateless and systematic nature of RaptorQ codes makes them ideal for multicasting data. A sender initially pushes a window of encoding symbols to all receivers, which then start pulling additional (source and repair) ones. A Polyraptor sender aggregates pull requests and multicasts a new symbol only after all receivers have sent one. As part of our current work is to be able to detect and eliminate straggler receivers by detaching them from the group and exchanging symbols with them independently through a one-to-one Polyraptor session.

3 DISCUSSION

In Figure 1a, we present Polyraptor's performance in a distributed storage scenario with 1 and 3 replicas. The three replica servers are randomly selected outside the client's rack. We have emulated the same behaviour with TCP by multi-unicasting data to the randomly selected servers. We have simulated 10,000 sessions (flows). Our multicasting model follows the design in [2]. Polyraptor maintains excellent performance when replicating data to 3 servers due to the underlying multicast support. Packet trimming along with RQ coding provide resilience against transient and persistent congestion. In order to demonstrate Polyraptor's performance when multi-sourcing data, we simulated a distributed storage scenario where a client fetches data from 1 and 3 replica servers at the same time. We emulated this behaviour with TCP by assuming that storage servers transfer back to the client part of the requested blocks without requiring any coordination. Figure 1b follows the same pattern as Figure 1a. Polyraptor sustains excellent performance fully utilising all available data replicas and the underlying network resources. Figure 1c presents a classic Incast scenario with synchronised short flows. Packet trimming along with the rateless nature of the RQ codes result in Incast elimination.

As part of our current work, we are evaluating Polyraptor's behaviour under different workloads and the existence of network hotspots. We are also looking at the influence of RQ encoding/decoding complexity, latency and decoding failure probability in the performance of Polyraptor.

REFERENCES

[1] C. Raiciu et al. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. Proc. of SIGCOMM.

[2] D. Li et al. 2014. Reliable Multicast in data center networks. IEEE Transactions on Computers.

[3] G.Parisis et al. 2013. Trevi: Watering Down Storage Hotspots with Cool Fountain Codes. Proc. of HotNets.

[4] K. Rashmi et al. 2013. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster. Proc. of USENIX.

[5] M.Luby et al. [n. d.]. RaptorQ Forward Error Correction Scheme for Object Delivery. IETF, RFC 6330, 2011.

[6] M. Handley et al. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. Proc. of SIGCOMM.

[7] M. Kheirkhah et al. 2016. MMPTCP: A multipath transport protocol for data centers. Proc. of INFOCOM.

[8] P. Cheng et al. 2014. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. Proc. of USENIX.

[9] S. Ghemawat et al. 2003. The Google File System. SOSP.

SCDP: Systematic Rateless Coding for Efficient Data Transport in Data Centres

Mohammed Alasmar*, George Parisis*, Jon Crowcroft †
*School of Engineering and Informatics, University of Sussex, UK, Email: {m.alasmar, g.parisis}@sussex.ac.uk
†Computer Laboratory, University of Cambridge, UK, Email: Jon.Crowcroft@cl.cam.ac.uk

***Abstract*—In this paper we propose SCDP, a novel, general-purpose data transport protocol for data centres that, in contrast to all other protocols proposed to date, natively supports one-to-many and many-to-one data communication, which is extremely common in modern data centres. SCDP does so without compromising on efficiency for short and long unicast flows. SCDP achieves this by integrating RaptorQ codes with receiver-driven data transport, in-network packet trimming and Multi-Level Feedback Queuing (MLFQ); (1) RaptorQ codes enable efficient one-to-many and many-to-one data transport; (2) on top of RaptorQ codes, receiver-driven flow control, in combination with in-network packet trimming, enable efficient usage of network resources as well as multi-path transport and packet spraying for all transport modes. Incast and Outcast are eliminated; (3) the systematic nature of RaptorQ codes, in combination with MLFQ, enable fast, decoding-free completion of short flows. We extensively evaluate SCDP in a wide range of simulated scenarios with realistic data centre workloads. For one-to-many and many-to-one transport sessions, SCDP performs significantly better compared to NDP. For short and long unicast flows, SCDP performs equally well or better compared to NDP.**

***Index Terms*—Data centre networking, data transport protocol, fountain coding, modern workloads.**

I. Introduction

Data centres support the provision of core Internet services and it is therefore crucial to have in place data transport mechanisms that ensure high performance for the diverse set of supported services. Data centres consist of a large number of commodity servers and switches, support multiple paths among servers, which can be multi-homed, very large aggregate bandwidth and very low latency communication with shallow buffers at the switches.

One-to-many and many-to-one communication. A significant portion of data traffic in modern data centres is produced by applications and services that replicate data for resilience purposes. For example, distributed storage systems, such as GFS/HDFS [1], [2] and Ceph [3], replicate data blocks across the data centre (with or without daisy chaining[1]). Partition-aggregate [4], [5], streaming telemetry [6]–[8], and distributed messaging [9], [10] applications also produce similar traffic workloads. Multicast has already been deployed in data centres[2] and, with the advent of P4, scalable multicasting is becoming practical [11]. As a result, much research on scalable network-layer multicasting in data centres has recently emerged [12]–[16].

[1]https://patents.google.com/patent/US20140215257

[2]e.g. https://www.rackspace.com/en-gb/cloud/networks

Existing data centre transport protocols are suboptimal in terms of network and server utilisation for these workloads. One-to-many data transport is implemented through multi-unicasting or daisy chaining for distributed storage. As a result, copies of the same data is transmitted multiple times, wasting network bandwidth and creating hotspots that severely hurt the performance of short, latency-sensitive flows.

In many application scenarios, multiple copies of the same data can be found in the network at the same time (e.g. in replicated distributed storage) but only one replica server is used to fetch it. Fetching data from all servers, in parallel, from all available replica servers (many-to-one data transport) would provide significant benefits in terms of eliminating hotspots and naturally balancing load among servers

These performance limitations are illustrated in Figure 1, where we plot the application goodput for TCP and NDP [17] in a distributed storage scenario with 1 and 3 replicas. When a single replica is stored in the data centre, NDP performs very well, as also demonstrated in [17]. TCP performs poorly[3]. On the other hand, when three replicas are stored in the network, both NDP and TCP perform poorly in both write and read workloads. Writing data involves either multi-unicasting replicas to all three servers (bottom two lines in Figure 1a) or daisy chaining replica servers (the line with the diamond marker); although daisy chaining performs better, avoiding the bottleneck at the client's uplink, they both consume excessive bandwidth by moving multiple copies of the same block in the data centre. Fetching a data block from a single server when it is stored in two more servers creates hotspots at servers' uplinks due to collisions from randomly selecting a replica server for each read request (see 3-sender goodput performance in Figure 1b).

Long and short flows. Modern cloud applications commonly have strict latency requirements [18]–[23]. At the same time, background services require high network utilisation [24]–[27]. A plethora of mechanisms and protocols have been proposed to date to provide efficient access to network resources to data centre applications, by exploiting support for multiple equal-cost paths between any two servers [17], [26], [28], [29] and hardware capable of low latency communication [22], [30], [31] and eliminating Incast [32]–[35] and Outcast [36]. Recent proposals commonly focus on a single dimension of

[3]It is well-established that TCP is ill-suited for meeting throughput and latency requirements of applications in data centre networks, therefore we will be using NDP [17] as the baseline protocol throughout this paper.

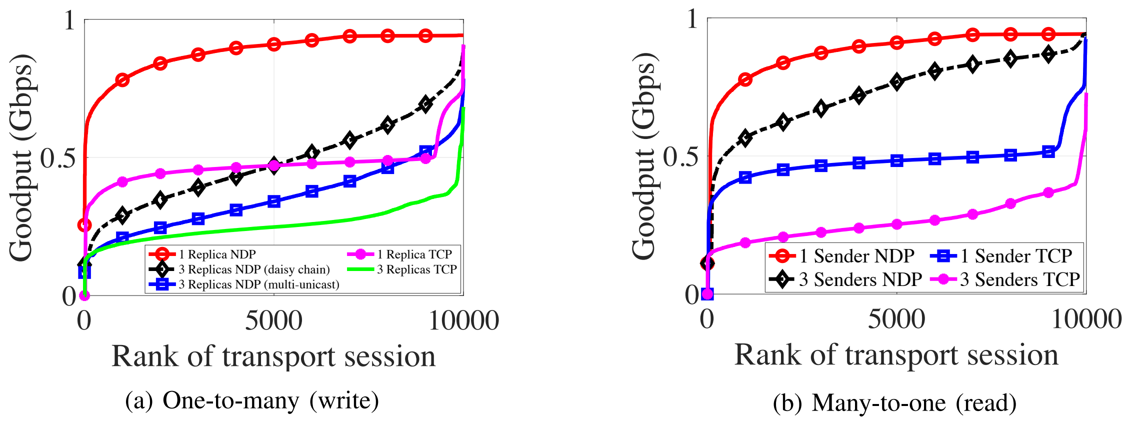


Fig. 1: Goodput results in a 250-server FatTree topology with 1GB link speed & 10μs link delay. Background traffic is present to simulate congestion. Results are for 10,000 (a) write and (b) read block requests (2MB each). Each I/O request is 'assigned' to a host in the network, which is selected uniformly at random and acts as the client. Requests' arrival times follow a Poisson process with $\lambda = 1000$. Replica selection and placement is based on HDFS' default policy (see Section IV-A for a full description of the experimental setup).

the otherwise complex problem space; e.g. TIMELY [37], DCQCN [38], QJUMP [39] and RDMA over Converged Ethernet v2 [40] focus on low latency communication but do not support multi-path routing. Other approaches [26], [27] do provide excellent performance for long flows but perform poorly for short flows [24], [28]. None of these protocols supports efficient one-to-many and many-to-one communication.

Contribution. In this paper we propose SCDP[4], a general-purpose transport protocol for data centres that, unlike any other protocol proposed to date, supports efficient one-to-many and many-to-one communication. This, in turn, results in significantly better overall network utilisation, minimising hotspots and providing more resources to long and short unicast flows. At the same time, SCDP supports fast completion of latency-sensitive flows and consistently high-bandwidth communication for long flows. SCDP eliminates Incast [32], [33], [35] and Outcast [36]. All these are made possible by integrating RaptorQ codes [43], [44] with receiver-driven data transport [17], [22], in-network packet trimming [17], [45] and Multi-Level Feedback Queuing (MLFQ) [46].

RaptorQ codes are systematic and rateless, induce minimal network overhead and support excellent encoding/decoding performance with low memory footprint (§II). They naturally enable one-to-many (§III-E) and many-to-one (§III-F) data transport. They support per-packet (encoded symbol) multi-path routing and multi-homed network topologies [47], [48] (§III-C); packet reordering does not affect SCDP's performance, in contrast to protocols like [18], [24], [28]. In combination with receiver-driven flow control (§III-D), and packet trimming (§III-C), SCDP eliminates Incast and Outcast, playing well with switches' shallow buffers. The systematic nature of RaptorQ codes enables fast, decoding-free completion of latency-sensitive flows by prioritising newly established ones, therefore eliminating loss (except under very heavy loads) (§III-H). Long flows are latency-insensitive so lost symbols can be recovered by repair ones; SCDP employs pipelining of source blocks, which alleviates the decoding overhead for large data blocks and maximises application goodput (§III-G). SCDP is a simple-to-tune protocol, which, as with NDP and scalable multicasting, will be deployable when P4 switches [49] are deployed in data centres.

[4]SCDP builds on our early work on integrating fountain coding in data transport protocols [41], [42].

SCDP performance overview. We found that SCDP improves goodput performance by up to ~50% compared to NDP with different application workloads involving one-to-many and many-to-one communication (§IV-A). Equally importantly, it reduces the average FCT for short flows by up to ~45% compared to NDP under two realistic data centre traffic workloads (§IV-B). For short flows, decoding latency is minimised by the combination of the systematic nature of RaptorQ codes and MLFQ; even in a 70% loaded network, decoding was needed for only 9.6% of short flows. This percentage was less than 1% in a 50% congested network (§IV-F). The network overhead induced by RaptorQ codes is negligible compared to the benefits of supporting one-to-many and many-to-one communication. Only 1% network overhead was introduced under a heavily congested network (§IV-G). RaptorQ codes have been shown to perform exceptionally well even on a single core, in terms of encoding/decoding rates. We therefore expect that with hardware offloading, in combination with SCDP's block pipelining mechanism (§III-G), the required computational overhead will be insignificant.

II. RaptorQ Encoding and Decoding

Encoding. RaptorQ codes are *rateless* and *systematic*. The input to the encoder is one or more *source blocks*; for each one of these source blocks, the encoder creates a potentially very large number of *encoding symbols* (rateless coding). All source symbols (i.e. the original fragments of a source block) are amongst the set of encoding symbols (systematic coding). All other symbols are called *repair* symbols. Senders initially send source symbols, followed by repair symbols, if needed.

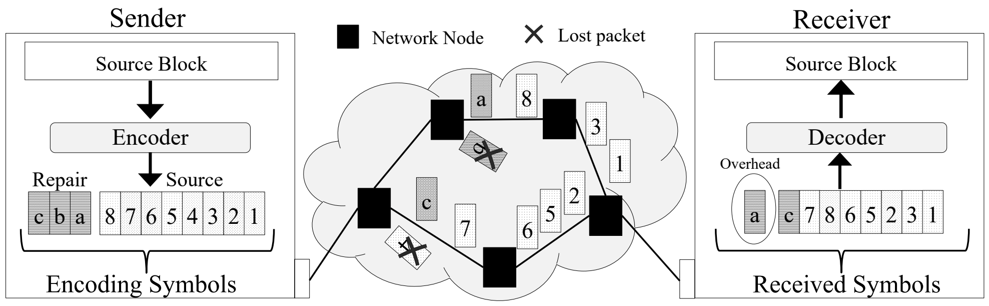


Fig. 2: RaptorQ-based communication

Decoding. The decoder decodes a source block after receiving a number of encoding symbols that must be equal to or larger than the number of source symbols; all symbols contribute to the decoding process equally. In a lossless communication scenario, decoding is not required, because all source symbols are available (systematic coding).

Performance. In the absence of loss, RaptorQ codes do not incur any network or computational overhead. The trade-off associated with RaptorQ codes when loss occurs is with respect to some (1) minimal network overhead to enable successful decoding of the original fragments and (2) computational overhead for decoding the received symbols to the original fragments. RaptorQ codes behave exceptionally well in both respects. With two extra encoding symbols (compared to the size of original fragments), the decoding failure probability is 10^{-6}. It is important to note that decoding failure is not fatal; instead more encoding symbols can be requested. The time complexity of RaptorQ encoding and decoding is linear to the number of source symbols. RaptorQ codes support excellent performance for all block sizes, including very small ones, which is very important for building a general-purpose data transport protocol that is able to handle equally efficiently different types of workloads. In [50], the authors report encoding and decoding speeds of over 10 Gbps using a RaptorQ software prototype running on a single core. With hardware offloading RaptorQ codes would be able to support data transport at line speeds in modern data centre deployments. On top of that, multiple blocks can be decoded in parallel, independently of each other. Decoding small source blocks is even faster, as reported in [51]. The decoding performance does not depend on the sequence that symbols arrived nor on which ones do.

Example. Before explaining in detail how RaptorQ codes are integrated in SCDP, we present a simple example of point-to-point communication between two hosts, which is illustrated in Figure 2.[5] On the sender side, a single source block is passed to the encoder that fragments it into K= 8 equal-sized source symbols $S_1, S_2, ..., S_8$. The encoder uses the source symbols to generate repair symbols $S_\text{a}, S_\text{b}, S_\text{c}$ (here, the decision to encode 3 repair symbols is arbitrary). Encoding symbols are transmitted to the network, along with the respective encoding symbol identifiers (ESI) and source block numbers (SBN) [43]. As shown in Figure 2, symbols S_4 and S_b are lost. Symbols take different paths in the network but this is transparent to the receiver that only needs to collect a specific amount of encoding symbols (source and/or repair). The receiver could have been receiving symbols from multiple senders through different network interfaces. In this example, the receiver attempts to decode the original source block upon receiving 9 symbols, i.e. one extra symbol which is a necessary network overhead (as shown in Figure 2). Decoding is successful and the source block is passed to the receiver application. As mentioned above, if no loss had occurred, there would be no need for decoding and the data would have been directly passed to the application.

[5]Note that Figure 2 does not illustrate SCDP's underlying mechanisms for requesting encoding symbols and flow control. It is only intended to showcase the main features of RaptorQ codes, which SCDP builds on. The design of SCDP is discussed extensively in Section III.

III. SCDP Design

In this section, we describe SCDP in detail. We first present an overview of the protocol and discuss its key design decisions. We define SCDP's packet types and switch service model, the supported communication modes and how efficiency is provided for short and long flows.

A. Design Overview

Figure 3 illustrates SCDP's key components (shown in rectangles) and how these are brought together to tackle the challenges identified in Section I (shown in ellipses). SCDP is a receiver-driven transport protocol, which allows for swift reactions to congestion when observing loss; more specifically trimmed headers, as discussed in Section III-B (*no Incast*, *no hotspots* in Figure 3). Initially, senders push a pre-specified number of symbol packets, starting with source symbols; subsequently, receivers request additional symbols at their link capacity (*no Incast*) until they can decode the respective source block. Transport sessions are initiated immediately, without any handshaking, by the source symbols pushed by the sender(s) (*fast FCT* in Figure 3). In SCDP, there are no explicit acknowledgements; a request for an encoding symbol implicitly acknowledges the reception of a symbol. SCDP adopts packet trimming to provide fast congestion feedback to receivers (*no Incast*, *no Outcast* in Figure 3) and MLFQ, as in [46], to eliminate losses for short flows (except under extreme congestion); this, along with the implicit connection establishment, results in fast, decoding-free completion of

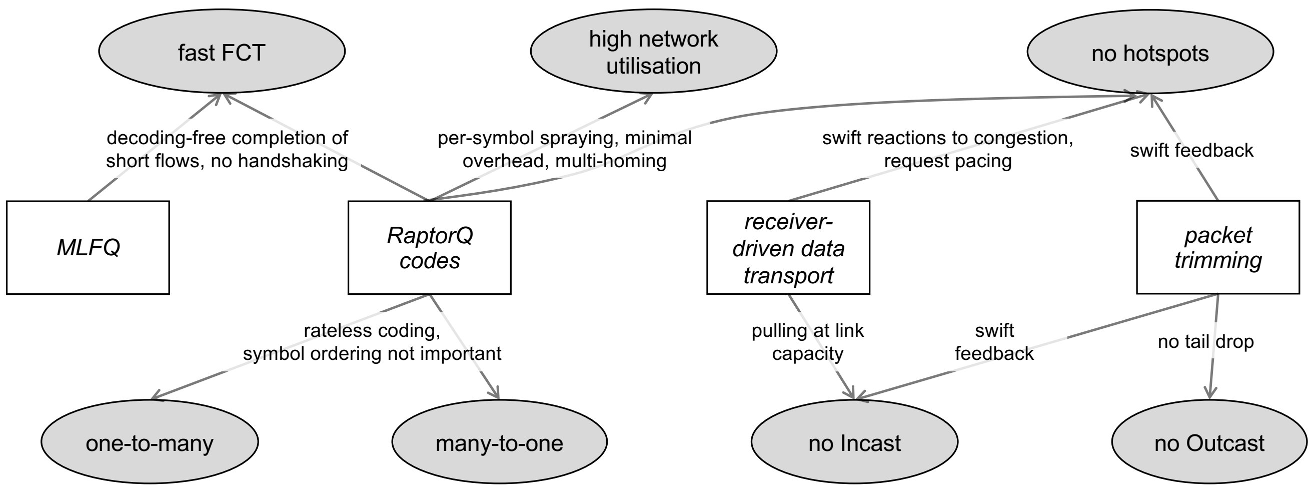

fast FCT
high network utilisation
no hotspots
decoding-free completion of short flows, no handshaking
per-symbol spraying, minimal overhead, multi-homing
swift reactions to congestion, request pacing
swift feedback
MLFQ
RaptorQ codes
receiver-driven data transport
packet trimming
rateless coding, symbol ordering not important
pulling at link capacity
swift feedback
no tail drop
one-to-many
many-to-one
no Incast
no Outcast

Fig. 3: SCDP's key components

(almost all) short flows (*fast FCT*). RaptorQ codes incur minimal network overhead due to the extra repair symbols when loss occurs, therefore network utilisation is not affected (*high network utilisation* in Figure 3).

A unique feature of SCDP that differentiates it from all previous proposals is its support for *one-to-many* (multicast) and *many-to-one* (multi-source) communication modes (see Figure 3). In SCDP's one-to-many communication mode, a sender initially pushes a window of symbols to all receivers, which then start pulling additional (source and/or repair) ones. Senders aggregate pull requests and multicast a new symbol after receiving a pull request from all receivers. In the many-to-one mode, a receiver pulls encoding symbols from multiple senders. Duplicate symbols are avoided by having senders partitioning the stream of repair symbols in a distributed fashion [43]. Senders initially select and send a subset of source symbols, before sending repair symbols. Multi-source transport enables natural load balancing; (1) at the server level, each server contributes symbols at its available capacity; (2) at the network level, more symbols come through less congested paths. Unicast transport is a specialisation of many-to-one transport with one sender.

In SCDP, receivers are oblivious of the provenance of encoding symbols. Symbols can follow different paths in the network, enabling per-packet ECMP routing. Symbol reordering does not affect decoding performance. Symbols can also be received from different network interfaces, enabling multi-homed communication [47], [48] (*high network utilisation*).

B. Packet Types

SCDP supports three types of packets. A *symbol* packet carries one MTU-sized encoding symbol, either source or repair, the respective source block number (SBN) (i.e. the source block it belongs to), and the encoding symbol identifier (ESI), which identifies a symbol within a stream of source and repair symbols for a specific source block [43]. Data packets also include port numbers for identifying transport sessions, a priority field that is set by the sender and a *syn* flag that is set for all symbol packets that senders initially push.

A *pull* packet is sent by a receiver to request a symbol and contains a sequence number and a *fin* flag. Note that multiple symbol packets may be sent in response to a single pull request, as described in Section III-D. The sequence number is only used to indicate to a sender how many symbols to send (e.g. if pull requests get reordered due to packet spraying in the network). [6]The *fin* flag is used to identify the last pull request; upon receiving such a pull request, a sender sends the last symbol packet for this SCDP session.

Header packets are trimmed versions of symbol packets. Whenever a network switch receives a symbol packet that cannot be buffered, instead of dropping it, it trims its payload (i.e. a source or repair RaptorQ symbol) and forwards the remaining header with the highest priority. Header packets are a signal of congestion and are used by receivers for flow control and to always keep a window worth of symbol packets on the fly.

C. Switch Service Model

SCDP relies on network switching functionality that is either readily available in today's data centre networks [22] or is expected to be [17] when P4 switches are widely deployed. Note that it does not require any more switch functionality than NDP [17], QJUMP [39], or PIAS [46] do.

Priority scheduling and packet trimming. In order to support latency-sensitive flows, we employ MLFQ [46], and packet trimming [45]. We assume that network switches support a small number of queues (with respective priority levels). The top priority queue is only used for header and pull packets. This is crucial for swiftly providing feedback to receivers about loss in the network. Given that both types of packets are very small, it is extremely unlikely that the respective

[6]Note that RaptorQ codes are rateless, therefore there is no need for receivers to request lost symbols; a new symbol will equally contribute to the decoding of the source block.

queue gets full and that they are dropped[7]. The rest of the queues are very short and are used to buffer symbol packets. Switches perform weighted round-robin scheduling between the top-priority (header/pull) queue and the symbol packet queues. This guards against congestion collapse, a situation where a switch only forwards trimmed headers and all symbol packets are trimmed (to headers). When a data packet is to be transmitted, the switch selects the head packet from the highest priority, non-empty queue. In combination with the priority setting mechanism, this minimises loss for short flows, enabling fast, decoding-free completion.

Multipath routing. SCDP packets are sprayed to all available equal-cost paths to the destination[8] in the network. SCDP relies on ECMP and spraying could be done either by using randomised source ports [24], or the ESI of symbol and header packets and the sequence number of pull packets.

D. Unicast Transport Sessions

A unicast SCDP transport session is implicitly opened by a sender by pushing a window of symbol packets to the receiver. Senders tag outgoing symbol packets with a priority value, which is used by the switches when scheduling their transmission (§III-C). The priority of outgoing symbol packets is gradually degraded, when specific thresholds are reached. Calculating these thresholds can be done as in PIAS [46] or AuTO [30]. After receiving the initial window of packets, the receiver takes control of the flow of incoming packets by pacing pull requests to the sender. A pull request carries a sequence number which is auto-incremented for each incoming symbol packet. The sender keeps track of the sequence number of the last pull request and, upon receiving a new pull request, it will send one or more packets to fill the gap between the sequence numbers of the last and current request. Such gaps may appear when pull requests are reordered due to packet spraying. Senders ignore pull requests with sequence numbers that have already been ‘served’; i.e. when they had previously responded to the respective pull requests.

Receivers maintain a single queue of pull requests for all active transport sessions. Flow control’s objective is to keep the receiver’s incoming link as fully utilised as possible at all times. This dictates the pace at which receivers send pull requests to all different senders. Receivers buffer encoding symbols along with their ESI and SBN and start decoding a source block upon receiving either K source symbols, where K is the total number of source symbols, or $K+o$ source and repair symbols, when loss occurs (o is the induced network overhead). We found that $o = 2$ extra symbols, when loss occurs, is the sweet spot with respect to the overhead and decoding failure probability trade-off.

The receiver sets the *fin* flag in the pull request for the last symbol (a source or repair symbol at that point) that sends to the sender. Note that this may not actually be the last request the receiver sends, because the symbol packet that is sent in response to that request may get trimmed. All pull requests for the last required symbol (not a specific one) are sent with the *fin* flag on. The sender responds to fin-enabled pull requests by sending the next symbol in the potentially very large stream of source and repair symbols with the highest priority. It finally releases the transport session only after a time period that ensures that the last prioritised symbol packet was not trimmed. This time period is very short; in the very unlikely case that the prioritised symbol packet was trimmed, the respective header would be prioritised along with the pull packet subsequently sent by the receiver.

E. One-to-many Transport Sessions

One-to-many transport sessions exploit support for network-layer multicast (e.g. with [11]–[16]) and coordination at the application layer; for example, in a distributed storage scenario, multicast groups could be pre-established for different replica server groups or setup on demand by a metadata storage server. This would eliminate the associated latency overhead for establishing multicast groups on the fly and is practical for other data centre multicast workloads, such as streaming telemetry and distributed messaging, where destination servers are known at deployment time. With recent advances in scalable data centre multicasting, a very large number of multicast groups can be deployed with manageable overhead in terms of switch state and packet size. For example, Elmo [11] encodes multicast group information inside packets, therefore minimising the need to store state at the network switches. With small group sizes, as in the common data centre use cases mentioned above, Elmo can support an extremely large number of groups, which can be encoded directly in packets, eliminating any maintenance overhead associated with churn in the multicast state. “In a three-tier data centre topology with 27K hosts, Elmo supports a million multicast groups using a 325-byte packet header, requiring as few as 1.1K multicast group-table entries on average in leaf switches, with a traffic overhead as low as 5% over ideal multicast” [11].

As with unicast transport sessions, an SCDP sender initially pushes IW (*syn*-enabled) symbol packets tagged with the highest priority. Receivers then request more symbols by sending respective pull packets. The sender sends a new symbol packet only after receiving a request from all receivers within the same multicast group. Receivers queue and pace pull packets as in all other transport modes. Depending on the network conditions and server load, a receiver may get behind in terms of received symbols. The rateless property of RaptorQ codes is ideal for such situation; within a single transport session, receivers may receive a different set of symbols but they will all decode the original source block as long as the required number of symbols is collected, regardless of which symbols they missed (see Section II). On the other hand, some receivers may end up receiving more symbols than what would be required to decode the original source block. This is unnecessary network overhead induced by SCDP but, in Section IV-G, we show that even under severe congestion, SCDP performs significantly better than NDP, exploiting the support for network-layer multicast. In

[7]Receivers employ a simple timeout mechanism, as in [17], to recover from the unlikely losses of pull and header packets.

[8]In SCDP’s one-to-many communication mode there are many destinations. In Section III-E, we describe this communication mode in detail.

extreme scenarios where receivers become unresponsive, this overhead increases significantly, as all other receivers will be unnecessarily receiving a potentially very large number of symbols. In such situations, detaching the straggler server from the multicast group (at the application layer) would trivially solve the issue.

F. Many-to-one Transport Sessions

Many-to-one data transport is a generalisation of the unicast transport discussed in Section III-D. Senders initialise a multi-source transport session by pushing an initial window IW_i of symbol packets to the receiver. As in the unicast transport mode, these symbol packets have the *syn* flag set, are tagged with the highest priority and contain source or repair symbols. The total number of initially pushed symbol packets $IW_{total} = \sum_{i=1}^{n_s} IW_i$, where n_s is the total number of senders, is selected to be larger than the initial window IW used in unicast transport sessions. This is to enable natural load balancing in the data centre in the presence of slow senders or hotspots in the network. In that case, SCDP ensures that a subset of senders (e.g. 2 out of 3 in a 3-replica scenario) can still fill the receiver's downstream link. In Section IV-E, we show that initial window sizes that are greater than 10 symbol packets result in the same (high) goodput performance. A large initial window would inevitably result in more trimmed symbol packets, which however would not affect short, latency-sensitive flows that would always be prioritised over longer multi-source sessions.

As discussed in Section II, RaptorQ codes are rateless and all symbols contribute equally to the decoding process, therefore the receiver is agnostic to the origin of each symbol. In many-to-one communication scenarios, senders are coordinated at the application layer. For example, in a distributed storage scenario, clients can either resolve the IP addresses of servers in a deterministic way (e.g. as in [3], [52]) or by asking a metadata server (e.g. as in [53]). Before fetching the data, they are aware of (1) the total number of senders n_s and (2) the server index i in the set of all senders. As a result, they can partition the potentially large stream of source and repair (if needed) symbols so that each one produces unique symbol packets.

G. Maximising Goodput for Long Flows through Source Block Pipelining

With RaptorQ codes, if loss occurs, the receiver must decode the source block after collecting the required number of source and repair symbols (§II). This induces latency before the data can become available to the application.

For large source blocks, SCDP masks this latency by splitting the large source block to many smaller blocks, instead of encoding and decoding the whole block. The smaller blocks are then pipelined over a single SCDP session. With pipelining, a receiver decodes each one of these smaller source blocks while receiving symbol packets for the next one, effectively masking the latency induced by decoding, except for the last source block. The latency for decoding this last smaller block is considerably smaller compared to decoding the whole block at once.[9] For short, latency-sensitive flows, this could be a serious issue, but SCDP strives to eliminate losses, resulting in fast, decoding-free completion of short flows (§III-H).

H. Minimising Network Overhead and Completion Time for Short Flows

SCDP ensures that a window of IW symbol packets are on the fly throughout the lifetime of a transport session. The window decreases by one symbol packet for the last IW packets that the sender sends. As long as no loss is detected (through receiving a trimmed header), a receiver sends $K - IW$ pull requests, in total. For every received trimmed header (i.e. observed loss), the receiver sends a pull request, and, subsequently, the sender sends a new symbol, which equally contributes to the decoding of the source block. This ensures that SCDP does not induce any unnecessary overhead; i.e. symbol packets that are unnecessary for the decoding of the respective source block. The target for the total number of received symbols also changes when loss is detected. Initially, all receivers aim at receiving K source symbols. Upon receiving the first trimmed header, the target changes to $K + 2$, which ensures that decoding failure is extremely unlikely to occur (see Section II).

By prioritising earlier packets of a session over later ones through MLFQ, SCDP minimises loss for short flows. This has an extremely important corollary in terms of SCDP's computational cost; no decoding is required for the great majority of short flows, therefore completion times are almost always near-optimal. We extensively evaluate this aspect of SCDP's design in Section IV-F. It is important to note that for all supported types of communication, there is no latency induced due to encoding, because repair symbols can be generated while source symbols are sent; i.e. there can always be one or more repair symbols ready before they are needed.

IV. Experimental Evaluation

We have extensively evaluated SCDP's performance through large scale, packet-level simulations. We have developed models of SCDP, NDP, the switch service model and network-layer multicast support [54] in OMNeT++[10], [11]. Our results are fully reproducible. For our experimentation we have used a 250-server FatTree topology with 25 core switches and 5 aggregation switches in each pod (50 aggregation switches in total). This is a typical size for a simulated data centre topology, also used in the evaluation of recent data centre transport proposals [22], [23], [31], [46]. The values for the link capacity, link delay and switch buffer size are 1 Gbps, 10μs and 20 packets, respectively. The buffer is allocated to 5 packet queues with different scheduling priorities. The thresholds for demoting the priority for a specific session are statically assigned to 10KBs, 100KBs, 1MB and 10MBs,

[9]For the experimental evaluation presented in Section IV, we have integrated pipelining into the developed SCDP model and simulated the respective latency following the results reported in [51].

[10]Some of our models that we use in this paper have been published at OMNeT++ Community Summit [55].

[11]https://github.com/mohammedalasmar/ndpTcpDatacentreOmnetppModel

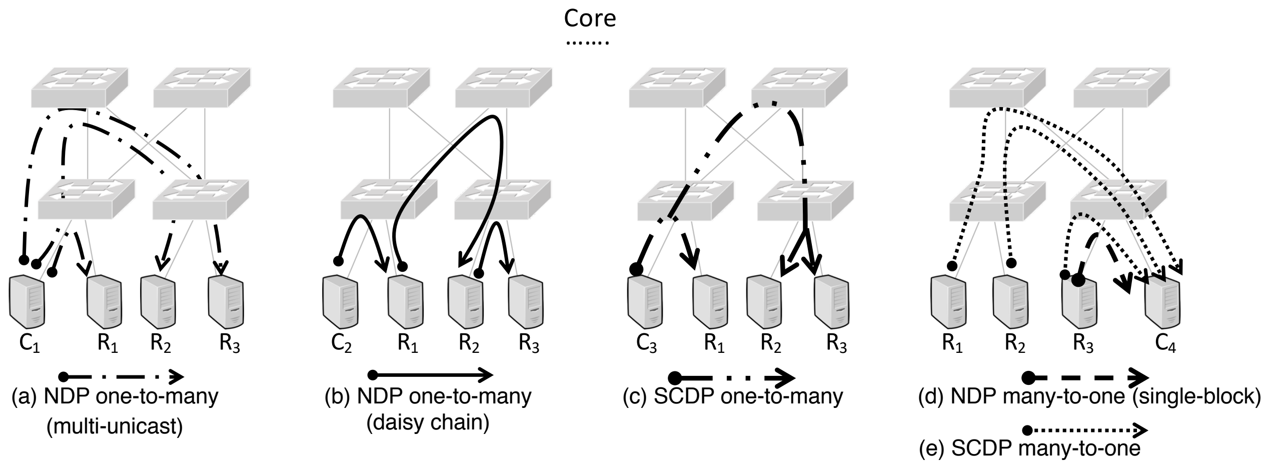

Core
C1
R1
R2
R3
C2
R1
R2
R3
C3
R1
R2
R3
R1
R2
R3
C4
(a) NDP one-to-many (multi-unicast)
(b) NDP one-to-many (daisy chain)
(c) SCDP one-to-many
(d) NDP many-to-one (single-block)
(e) SCDP many-to-one

Fig. 4: Illustration of read and write workloads and replica placement policy used in comparing goodput performance for SCDP and NDP. For clarity, the core of the data centre is omitted and replica groups (and the respective transport sessions) are selected to be in the same pod. In our simulations, the selection of remote racks to store data blocks is random and racks in different pods can be selected. Network-layer multicast is supported in SCDP one-to-many communication.

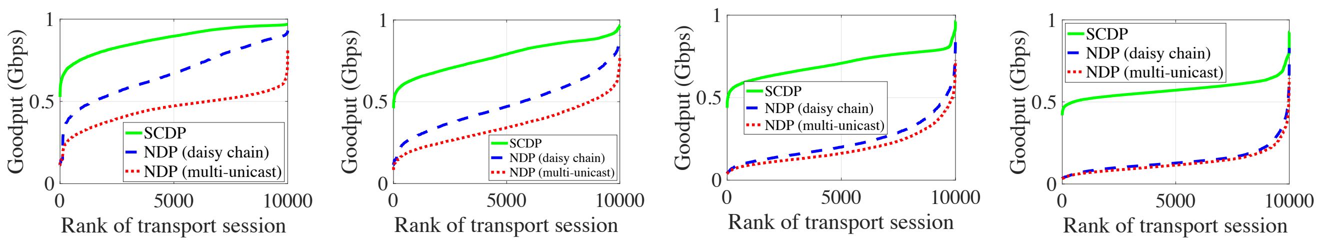
respectively[12]. The top priority queue is for pull and header packets which are very small, therefore we can avoid timeouts by setting its size to a relatively large value (as also done in [17]). Unless otherwise stated, the initial window IW for one-to-one and one-to-many sessions is set to 12 symbol packets. For many-to-one sessions the initial window is set to 6 symbol packets per sender. For all experiments we set the block size for pipelining to 100 MTU-sized symbol packets. We have run each simulation 5 times with different seeds and report average (with 95% confidence intervals) or aggregate values.

A. Goodput for One-to-Many and Many-To-One Communication

In this section we measure the application goodput for SCDP and NDP in a distributed storage setup with 3 replicas (as depicted in Figure 4).The setup involves many-to-one and one-to-many communication. In each run, we simulate 2000 transport sessions (or I/O requests at the storage layer) with sizes 1MB and 4MB each (*rs* in the figures). Transport session arrival times follow a Poisson process; we have used different λ values (2000 and 4000) to assess the performance of the studied protocols under different loads. Each I/O request is 'assigned' to a host in the network (C_i in Figure 4), which is selected uniformly at random and acts as the client. Replica selection and placement is based on HDFS' default policy. More specifically, we assume that clients are not data nodes themselves, therefore a data block is placed on a randomly selected data node (R_i in Figure 4). One replica is stored on a node in a different remote rack, and the last replica is stored on a different node in the same remote rack. A client will read a block from a server located in the same rack, or a randomly selected one, if no replica is stored in the same rack. In order to simulate congestion in the core of the network, 30% of the nodes run background long flows, the scheduling of which is based on a permutation traffic matrix.

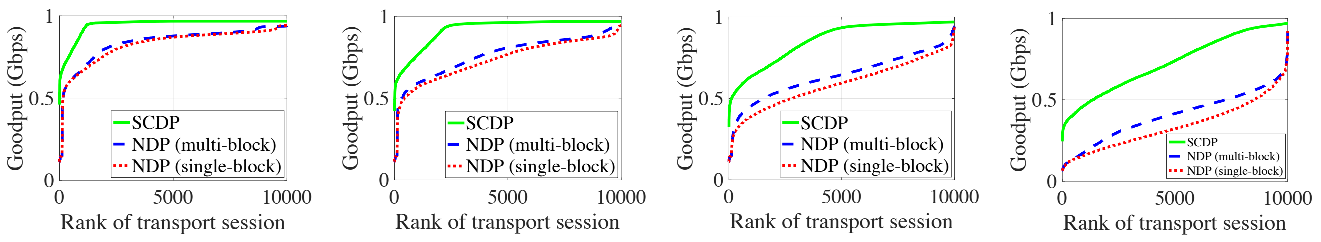
[12]In a real-world deployment these would be set dynamically, e.g. as in AuTO [30].

One-to-many transport sessions. We evaluate SCDP's performance in one-to-many traffic workloads and assess how it benefits from the underlying support for network-layer multicast, compared to NDP. One-to-many communication with NDP is implemented through (1) multi-unicasting data to multiple recipients (Figure 4a) or (2) daisy-chaining the transmission of replicas through the respective servers (Figure 4b). In daisy-chaining, each replica starts transmitting the data to the next replica server (according to HDFS's placement policy), as soon as it starts receiving data from another replica server. Daisy-chaining eliminates the bottleneck at the client's uplink. We measure the overall goodput from the time the client initiates the transmission until the last server receives the whole data. The results for various loads and I/O request sizes are shown in Figure 5. In all figures, flows are ranked according to the measured goodput performance (shown on the y axis). SCDP, with its natural load balancing and the support of multicast (Figure 4c), significantly outperforms NDP even when daisy-chaining is used for replicating data. Daisy-chaining is effective compared to multi-unicasting when the network is not heavily loaded. In SCDP, around 50% of the sessions experience goodput that is over 90% of the available bandwidth for 1MB sessions and $\lambda = 2000$. The remaining 50% sessions still get a goodput performance over 60% of the available bandwidth. When the network load is heavier, daisy-chaining does not provide any significant benefits over multi-unicasting because data needs to be moved in the data centre multiple times and congestions gets severe. For $\lambda = 4000$ and 4MB sessions, NDP's performance is significantly worse for most sessions, whereas SCDP still offers an acceptable transport service to all sessions. SCDP fully exploits the support for network-layer multicasting providing superior performance to all storage clients because the required network bandwidth is minimised. Minimising the bandwidth requirements for



(a) rs = 1MB, λ = 2000 (b) rs = 1MB, λ = 4000 (c) rs = 4MB, λ = 2000 (d) rs = 4MB, λ = 4000

Fig. 5: Performance comparison for SCDP and NDP - write I/O with 3 replicas (one-to-many)



(a) rs = 1MB, λ = 2000 (b) rs = 1MB, λ = 4000 (c) rs = 4MB, λ = 2000 (d) rs = 4MB, λ = 4000

Fig. 6: Performance comparison for SCDP and NDP - read I/O with 3 replicas (many-to-one)

one-to-many flows that are extremely common in the data centre, makes space for regular short and long flows. For the experimental setup with the heaviest network load ($\lambda = 4000$ and 4MB sessions), we have measured the average goodput for SCDP background traffic to be 0.408 Gbps, compared to 0.252 Gbps for the respective NDP experiment[13]. This is 15.6% of the available bandwidth freed up for all other flows. We evaluate the positive effect that SCDP has with respect to network hotspots in Section IV-C.

Many-to-one transport sessions. In the many-to-one scenario (Figure 6), clients read previously stored data from the network. SCDP naturally balances this load according to servers' capacity and network congestion, as discussed in Section III-F (see Figure 4e). With NDP, clients read data either from a replica server located in the same rack or a randomly selected server, if there is no replica stored in the same rack. For NDP, we simulate both a single-block (see Figure 4d) and multi-block request workload. The latter enables parallelisation at the application layer (e.g. the read-ahead optimisation where a client reads multiple consecutive blocks under the assumption that they will soon be requested). Here, we simulate a 3-block read-ahead policy and measure the overall goodput from the time the I/O request is issued until all 3 blocks are fetched. To make the results as comparable to each other as possible, for the 3-block setup we use blocks the size of which is one third of the size of the single-block scenario (as reported in Figure 6). We do not include multi-block results for SCDP as they are almost identical to the single-block case, confirming the argument that it naturally distributes the load without any application-layer parallelisation.

In Figure 6 we observe that SCDP significantly outperforms NDP for all different request sizes and λ values. Even under heavy load, SCDP provides acceptable performance to all transport sessions. This is the result of (1) the natural and dynamic load balancing provided to SCDP's many-to-one sessions and (2) MLFQ; long background flows run at the lowest priority to boost the performance of shorter flows. Around 82% of the sessions experience goodput that is above 90% of the available bandwidth for 1MB sessions and $\lambda = 2000$. In contrast, NDP offers this good performance to only 10% of the sessions. For $\lambda = 4000$ and 4MB sessions, NDP's performance is significantly worse for most sessions, whereas SCDP still offers good performance to all sessions. Notably, the performance difference between SCDP and NDP increases with the congestion in the network, with SCDP being able to provide acceptable levels of performance where NDP would not (e.g. in the presence of hotspots or in over-subscribed networks).

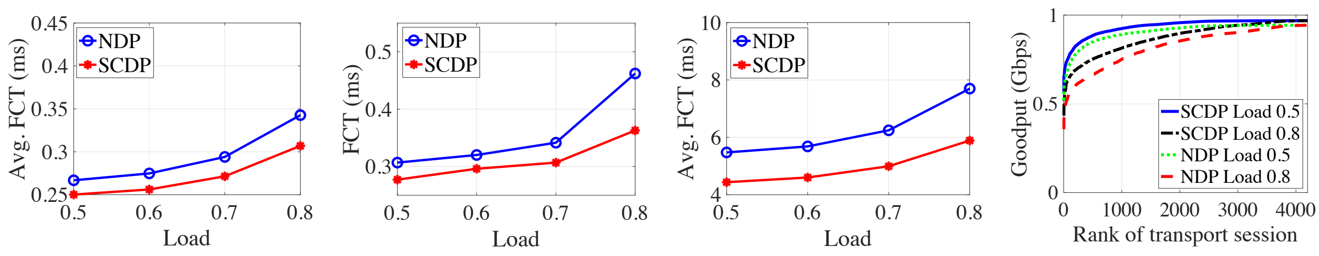
	0 - 10KB	10KB - 100KB	100KB - 1MB	1M- ..	Average flow size
Web Search [18]	49%	3%	18%	20%	1.6MB
Data Mining [56]	78%	5%	8%	9%	7.4MB

TABLE I: Flow size distribution of realistic workloads

B. *Performance Benchmarking with Realistic Workloads*

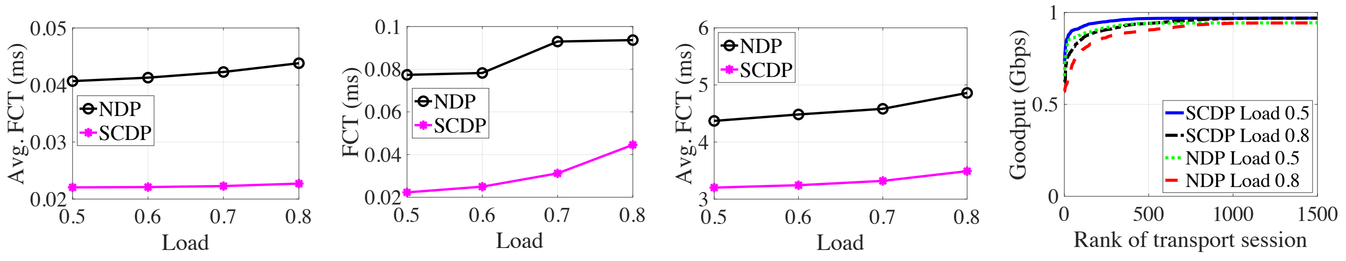
SCDP is a general-purpose transport protocol for data centres therefore it is crucial that it provides high performance for all supported transport modes and traffic workloads. In this section, we use realistic workloads reported by data centre operators to evaluate SCDP's applicability and effectiveness beyond one-to-many and many-to-one sessions. Here, we consider two typical services; *web search* and *data mining*

[13]Note that this improvement for background flows is despite these running at the lowest possible priority, given that they span the whole duration of the simulation.



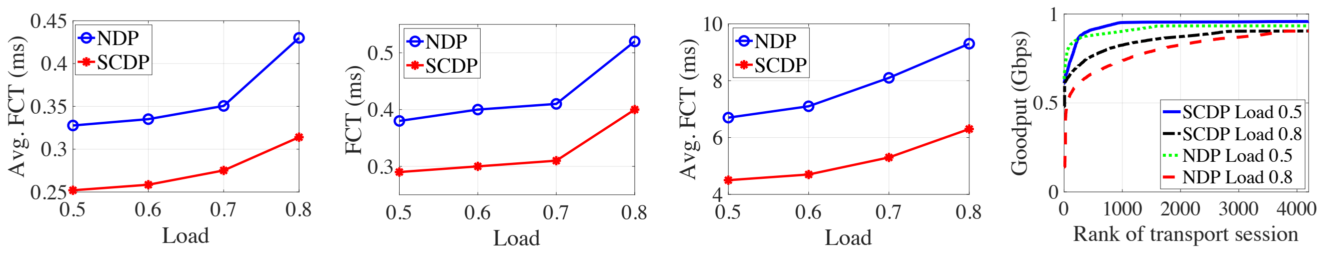
(a) (0, 100KB] (b) (0, 100KB] 99th percentile (c) (100KB, 1MB] (d) (1MB, 10MB]

Fig. 7: Web search workload with unicast flows as background traffic



(a) (0, 100KB] (b) (0, 100KB] 99th percentile (c) (100KB, 1MB] (d) (1MB, 10MB]

Fig. 8: Data mining workload with unicast flows as background traffic



(a) (0, 100KB] (b) (0, 100KB] 99th percentile (c) (100KB, 1MB] (d) (1MB, 10MB]

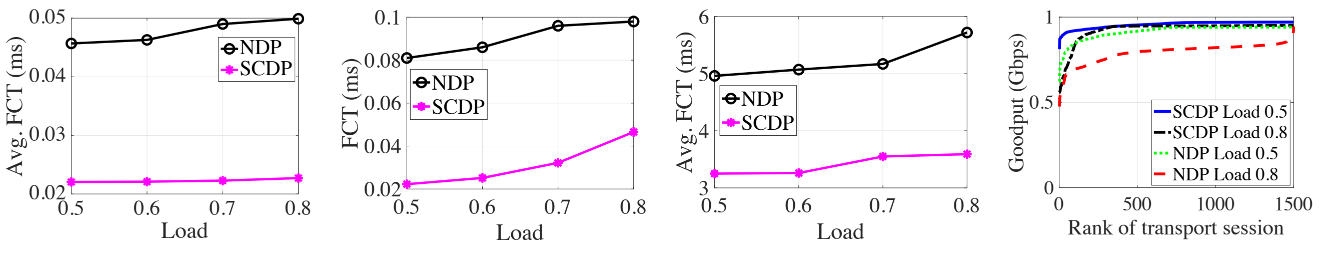
Fig. 9: Web search workload with a mixture of one-to-many and many-to-one sessions as background traffic

[18], [56]. The respective flow size distributions are shown in Table I. They are both heavy-tailed; i.e. a small fraction of long flows contribute most of the traffic. We have chosen the workloads to cover a wide range of average flow sizes ranging from 64KB to 7.4MB. We simulate four target loads of background traffic (0.5, 0.6, 0.7 and 0.8). We generate 20000 transport sessions, the inter-arrival time of which follows a Poisson process with $\lambda = 2500$. In Figures 7a and 7c and 8a and 8c, we report the average flow completion time (FCT) of flows with sizes in $(0 - 1\text{MB})$. For the shortest flows $(0-100\text{KB})$ we also report the 99th percentile of the measured FCTs (Figures 7b and 8b). Finally, Figures 7d and 8d illustrate the measured goodput for flows with sizes in (1MB, 10MB] (for load values of 0.5 and 0.8).

SCDP performs better in all scenarios due to the decoding-free completion of (almost all) short flows and the supported MLFQ. Note that when loss occurs, SCDP sessions must exchange 2 additional symbols; they also pay the 'decoding latency' price. For very short flows, the 99th percentile FCT is close to the average one for all loads, which indicates that this is rarely happening. We study the extent that this overhead and the associated decoding latency is required in Section IV-F. For higher loads, NDP performs even worse than SCDP because of the lack of support for MLFQ, which results in the trimming of more packets belonging to short flows. Note that the FCT of short flows in web search is larger than in data mining. This is mainly because the percentage of long flows in the former workload is larger than in the latter, resulting in a higher overall load (for all fixed loads of background traffic). A key message here is that SCDP provides significantly better tail performance for short flows compared to NDP, especially as the network load increases, despite the (very unlikely) potential for decoding and network overhead. For flows with sizes in (1MB, 10MB], we observe that goodput with SCDP is better compared to NDP; tail performance is also better.

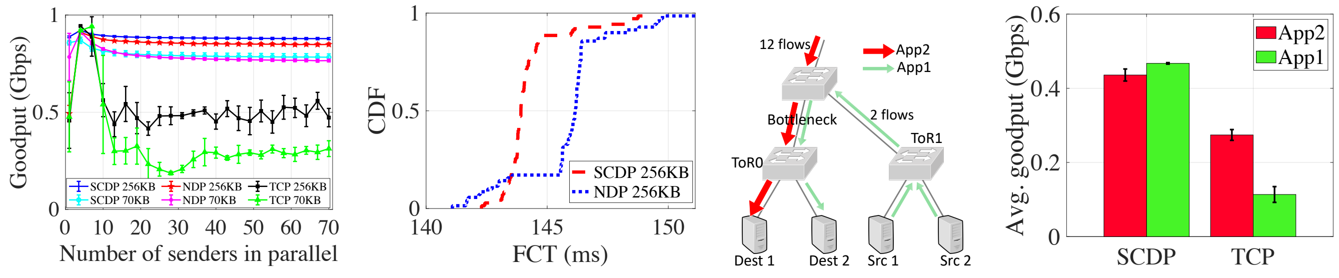
C. *Minimising Hotspots in the Network*

SCDP increases network utilisation by exploiting support for network-layer multicasting and enabling load balancing when data is fetched simultaneously from multiple servers, as demonstrated in Section IV-A. This, in turn, makes space in



(a) (0, 100KB] (b) (0, 100KB] 99th percentile (c) (100KB, 1MB] (d) (1MB, 10MB]

Fig. 10: Data mining workload with a mixture of one-to-many and many-to-one sessions as background traffic



(a) Incast: goodput comparison (b) Incast: FCT with 70 senders (c) Outcast: setup (d) Outcast: goodput

Fig. 11: Incast and Outcast evaluation

the network for regular short and long flows. In this section, we evaluate this performance benefit. We use as background traffic a 50%/50% mixture of write and read I/O requests (4MB each) that produce one-to-many and many-to-one traffic, respectively. We repeat the experiment of the previous section and evaluate the performance benefits of SCDP over NDP with respect to minimising hotspots and maximising network utilisation for regular short and long flows.

In Figures 9a and 9c, we observe that SCDP's performance is almost identical to the one reported in Figures 7a and 7c (similarly between Figure 8 and Figure 10). In contrast, NDP's performance deteriorates significantly because the background traffic requires more bandwidth (one-to-many) and results in hotspots at servers' uplinks (many-to-one). Tail performance for SCDP gets only marginally worse (the 99th percentile increases from 0.277ms to 0.287ms for the web search workload in load 0.5), whereas NDP's performance gets significantly worse (the 99th percentile increases from 0.306ms to 0.381ms). The observed behaviour is more pronounced in the web search workload which, as described in the previous section, results in higher overall network utilisation compared to the data mining workload.

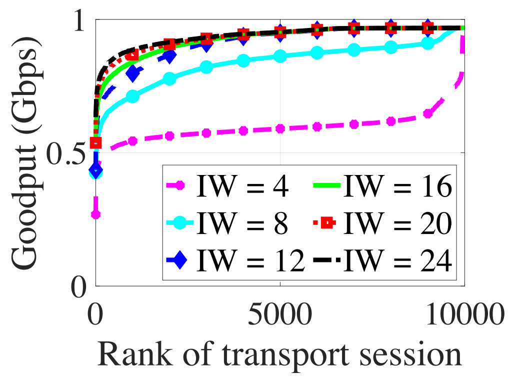
D. Eliminating Incast and Outcast

SCDP eliminates Incast by integrating packet trimming and not relying on retransmissions of lost packets (given the rateless nature of RaptorQ codes). We have simulated Incast by having multiple senders (ranging from 1 to 70) sending blocks of data (70KB and 256KB, each, in two separate experiments) to a single receiver. All transport sessions were synchronised and background traffic was present to simulate congestion. Figure 11a illustrates the measured aggregated goodput for all SCDP, NDP and TCP flows. Error bars represent the 95% confidence interval. As expected, TCP's performance collapses when the number of senders increases. SCDP performs slightly better compared to NDP even when a large number of servers send data to the receiver at the same time. This is attributed to the decoding-free completion of these flows, in combination with the packet trimming and the lack of retransmissions for SCDP. Figure 11b shows the CDF of the FCTs in the presence of Incast with 70 senders. We observe that for the vast majority of transport sessions, SCDP provides superior performance compared to NDP.

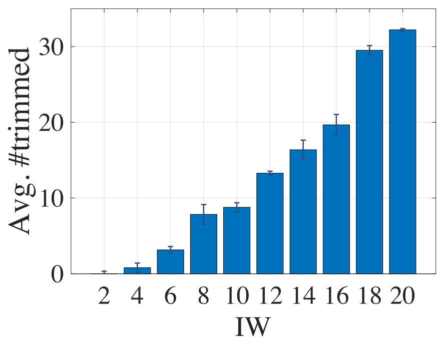
SCDP eliminates outcast by employing receiver-driven flow control and packet trimming, which prevent port blackout. We have simulated a classic outcast scenario, where two receivers that are connected to the same ToR switch receive traffic from senders located in the same pod (2 flows crossing 4 hops) and different pods (12 flows crossing 6 hops), respectively. Flow size is 200KB and all flows start at the same time. This is illustrated in Figure 11c. Here, the bottleneck link lies between the aggregate switch and the ToR switch, which is different from the Incast setup. Figure 11d shows the aggregate goodput for the two groups of flows, for SCDP and TCP. TCP Outcast manifests itself through (1) unfair sharing of the bottleneck bandwidth (around 113 and 274 Mbps for the groups of flows, respectively) and (2) suboptimal overall performance (around 0.387 Gbps). SCDP eliminates Outcast as the bottleneck is shared fairly between the two groups of flows (around 460 and 435 Mbps for the groups of flows, respectively, and the overall goodput is around 0.9 Gbps).

E. The effect of the Initial Window Size

A key parameter of SCDP is the initial windows IW of symbol packets that a sender pushes to the network.



(a) Goodput performance



(b) Number of trimmed packets

Fig. 12: The effect of the IW value

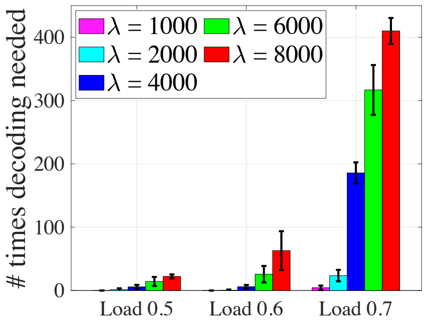


Fig. 13: Number of sessions that needed decoding for different loads and session inter-arrival times. The total number of simulated sessions is 5000

Throughout the lifetime of a transport session this window is maintained and only decreased for the last IW pull packets. In many-to-one transport sessions the sum of all the initial windows for all senders is set to be larger than the initial window for the one-to-one and one-to-many modes. This is to enable natural load balancing between all senders in the presence of congestion in the network (see Section III-F). In this section we evaluate the effect that the initial window has in the performance of SCDP. The experimental setup is as described in Section IV-A, with 1.5MB unicast sessions (we evaluated one-to-many and many-to-one sessions as well, which showed similar results as the unicast sessions).

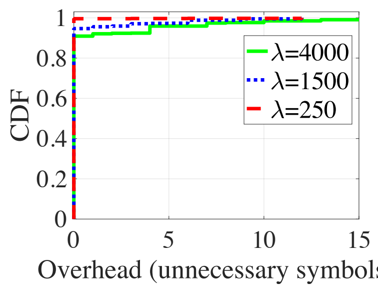
In Figure 12a, we observe that for very small values of the initial window, the goodput is very low and the receiver's downlink underutilised. As the window increases, utilisation approaches the maximum available link capacity (for 10 symbol packets). For larger values, the measured goodput is the same (full link capacity). This means that for many-to-one sessions, increasing the sum of initial windows for all senders does not have any negative impact on the goodput. However, increasing the window inevitably leads to more trimmed packets due to the added network load, which would be beyond the receiver's downlink capacity. This is illustrated in Figure 12b, where the average number of trimmed packets for session sizes of 1.5MB grows from 13 for an initial window of 12 symbol packets to 32 for an initial window of 20. This increase and the resulting necessity for decoding (and extra overhead) does not negatively affect many-to-one sessions which are commonly not latency-sensitive.

F. Network Overhead and Induced Decoding Latency

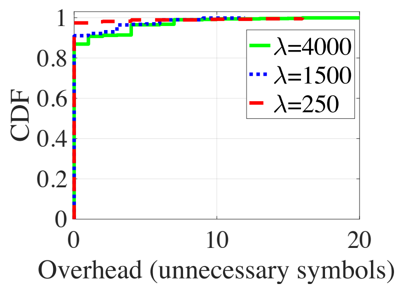
SCDP provides zero-overhead data transport when no loss occurs. In the opposite case, 2 extra symbols (compared to the number of original fragments) are required by the decoder to decode the source block (with extremely high probability). Additionally, the required decoding induces latency in receiving the original source block. Short flows in data centres are commonly latency sensitive so SCDP must be able to provide decoding-free completion of such flows. To asses the efficacy of our MLFQ-based approach, we measure the number of unicast flows that suffer symbol packet loss for different network loads ranging from 0.5 to 0.7. For each network load, we examine different λ values for the Poisson inter-arrival rate of the studied short flows (150KB). In each simulation, we generate 5000 sessions with the respective λ value as their inter-arrival time. In Figure 13, we observe that for load values of 0.5 and 0.6, the times that a short flow would require decoding and extra 2 symbol packets is very small (0.44% and 1.2% of the flows, respectively, when λ = 8000), rendering the respective overhead negligible.

G. Overhead in One-to-Many Sessions

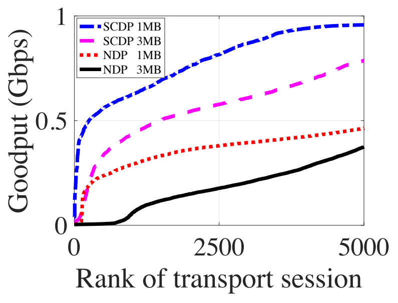
In Section III-E, we identified a limitation of SCDP with respect to unnecessary network overhead which may occur in one-to-many transport sessions in the presence of congestion. This is due to receivers getting behind with the reception of symbols. Consequently, up-to-date receivers will be receiving more symbols than what they actually need. In order to evaluate the extent of this limitation we setup a similar experiment to the one presented in Section IV-A. Figures 14a and 14b depict the CDF of the number of symbols that were sent unnecessarily for different values of λ, and session sizes. We observe that as the network load increases, the number of sessions that induce unnecessary network overhead increases. It is important to note that, even when this happens, the measured goodput for SCDP is significantly better than



(a) One-to-many - 1MB



(b) One-to-many - 3MB



(c) Goodput - 1MB and 3MB - $\lambda = 4000$

Fig. 14: Unnecessary network overhead in one-to-many sessions

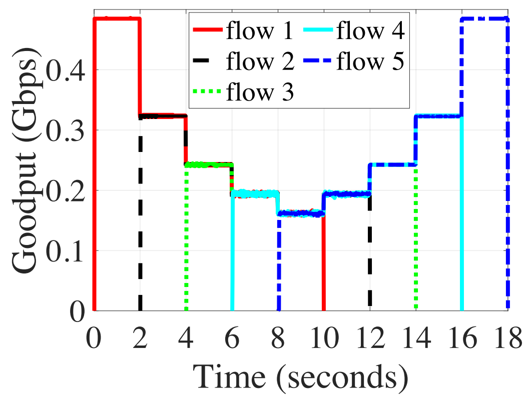


Fig. 15: Convergence test

that of NDP. Figure 14c illustrates the measured goodput for the examined session sizes and highest network load ($\lambda = 4000$). Clearly, SCDP significantly outperforms NDP despite the potential for some unnecessary network overhead. The benefit of exploiting network-layer multicast makes this potential overhead negligible.

H. Resource Sharing

SCDP achieves excellent fairness without needing additional mechanisms. SCDP's principles for resource sharing are as follows: (1) receivers pull symbol packets from one or more senders in the data centre at a pace that matches their downlink bandwidth. Given that servers are uniformly connected to the network with respect to link speeds, SCDP enables fair sharing of the network to servers. (2) A receiver will pull symbol packets for each SCDP session on a round robin basis.

As a result, SCDP enables fair sharing of its downlink to all transport sessions running at a specific receiver[14]. (3) SCDP employs MLFQ in the network. Obviously, this prioritisation scheme provides fairness between competing flows only within the same priority level. In Figure 15 we report goodput results with respect to the convergence behaviour of 5 SCDP unicast sessions that start sequentially with 2 seconds interval and 18 seconds duration, from 5 sending severs to the same receiving server under the same ToR switch. SCDP performs equally well to DCTCP in that respect [18]. Clearly, flows acquire a fair share of the available bandwidth very quickly. Each incoming flow is initially prioritised over the ongoing flows (MFLQ) but, given the reported time scales, this cannot be shown in Figure 15. We have repeated this experiment with larger number of flows, and we find that SCDP converges quickly, and all flows achieve their fair share.

[14]It would be straightforward to support priority scheduling at the receiver level as in NDP.

V. Conclusion

In this paper, we proposed SCDP, a general-purpose transport protocol for data centres that is the first to exploit network-layer multicast in the data centre and balance load across senders in many-to-one communication, while performing at least as well as the state of the art with respect to goodput and flow completion time for long and short unicast flows, respectively. Supporting one-to-many and many-to-one application workloads is very important given how extremely common they are in modern data centres [11]. SCDP achieves this remarkable combination by integrating systematic rateless coding with receiver-driven flow control, packet trimming and in-network priority scheduling.

RaptorQ codes incur some minimal network overhead, only when loss occurs in the network, but our experimental evaluation showed that this is negligible compared to the significant performance benefits of supporting one-to-many and many-to-one workloads. RaptorQ codes also incur computational overhead and associated latency when when loss occurs. However, we showed that this is rare for short flows because of MLFQ. For long flows, block pipelining alleviates the problem by splitting large blocks into smaller ones and decoding each of these smaller blocks while retrieving the next one. As a result, latency is incurred only for the last smaller block. RaptorQ codes have been shown to perform at line speeds even on a single core; we expect that with hardware offloading the overall overhead will not be significant.

In general and to the best of our knowledge, SCDP is the first protocol that provides native support for one-to-many and many-to-one communication in data centres and our extensive evaluation shows promising performance results.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP*, 2003.

[2] "HDFS architecture guide [Online] accessed Sep 2019," https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[3] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *USENIX*, 2006.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *USENIX OSDI'04*, 2004.

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI, USENIX*, 2012.

[6] "Open config. streaming telemetry." Accessed on 11/06/2019. [Online]. Available: http://blog.sflow.com/2016/06/streaming-telemetry.html

[7] "Microsoft Azure," Accessed on 11/06/2019. [Online]. Available: https://azure.microsoft.com/en-us/blog/cloud-service-fundamentals-telemetry-reporting/

[8] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," in *Elsevier Parallel Computing*, 2014.

[9] "Akka: Build powerful reactive, concurrent, and distributed applications more easily using udp," Accessed on 11/06/2019. [Online]. Available: https://doc.akka.io/docs/akka/2.5.4/java/io-udp.html

[10] "Jgroups: A toolkit for reliable messaging." Accessed on 11/06/2019. [Online]. Available: http://www.jgroups.org/overview.html

[11] M. Shahbaz, L. Suresh, N. Feamster, J. Rexford, O. Rottenstreich, and M. Hira, "Elmo: Source-routed multicast for cloud services," in *arxiv*, May 2018 [Online]Available:. [Online]. Available: http://arxiv.org/abs/1802.09815

[12] M. McBride and O. Komolafe, "Multicast in the data center overview," in *Huawei Arista Networks draft IETF*, 2019.

[13] D. Li, M. Xu, Y. Liu, X. Xie, Y. Cui, J. Wang, and G. Chen, "Reliable multicast in data center networks." IEEE Transactions on Computers, 2014.

[14] D. Li, J. Yu, J. Yu, and J. Wu, "Exploring efficient and scalable multicast routing in future data center networks." INFOCOM, 2011.

[15] W. Cui and C. Qian, "Dual-structure data center multicast using software defined networking," in *arxiv*, 2014. [Online]. Available: http://arxiv.org/abs/1403.8065

[16] X. Li and M. J. Freedman, "Scaling ip multicast on datacenter topologies," in *CoNEXT*, 2013.

[17] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. of SIGCOMM*, 2017.

[18] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)." Proc. of SIGCOMM, 2010.

[19] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *IEEE INFOCOM*, 2013.

[20] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *IEEE INFOCOM*, 2014.

[21] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen, "One more queue is enough: Minimizing flow completion time with explicit priority notification," in *IEEE INFOCOM*, 2017.

[22] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *In Proc. ACM SIGCOMM*, 2018.

[23] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proc. of SIGCOMM*, 2013.

[24] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. of INFOCOM*, 2016.

[25] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. of SIGCOMM*, 2011.

[26] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. of INFOCOM*, 2013.

[27] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks." Proc. of USENIX, 2010.

[28] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data Center Networking with Multipath TCP," in *Proc. of SIGCOMM*. ACM, 2010.

[29] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," *IEEE/ACM Transactions on Networking*, 2015.

[30] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization," in *In Proc. ACM SIGCOMM*, 2018.

[31] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed Near-optimal Datacenter Transport over Commodity Network Fabric," in *Proc. of CoNEXT*, 2015.

[32] Y. Chen, R. Griffith, J. Liu, and A. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. of SIGCOMM*, 2009.

[33] Y. Chen, R. Griffith, D. Zats, A. D. Joseph, and R. Katz, "Understanding TCP incast and its implications for big data workloads." Proc. of USENIX, 2012.

[34] C. Jiang, D. Li, and M. Xu, "LTTP: An LT-code based transport protocol for many-to-one communication in data centers," *IEEE Journal on Selected Areas in Communications*, 2014.

[35] C. J. Zheng, D. Li, M. Xu, and K., "A Coding-based Approach to Mitigate TCP Incast in Data Center Networks," *International Conference on Distributed Computing Systems Workshops*, 2012.

[36] P. Prakash, A. Dixit, Y. C. Hu, and R. Kompella, "The TCP Outcast Problem : Exposing Unfairness in Data Center Networks." Proc. of USENIX, 2012.

[37] R.Mittal, V.Lam, N.Dukkipati, E.Blem, H.Wassel, M.Ghobadi, A.Vahdat, Y.Wang, D.Wetherall, and D.Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *Proc. of SIGCOMM*, 2015.

[38] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *SIGCOMM*, 2015.

[39] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can JUMP them!" in *USENIX NSDI*, 2015.

[40] "Infiniband Trade Association. RoCEv2." 2014. [Online]. Available: https://cw.infinibandta.org/document/dl/7781

[41] M. Alasmar, G. Parisis, and J. Crowcroft, "Polyraptor: Embracing path and data redundancy in data centres for efficient data transport," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018.

[42] G. Parisis, T. Moncaster, A. Madhavapeddy, and J. Crowcroft, "Trevi: Watering Down Storage Hotspots with Cool Fountain Codes," in *Proc. of HotNets*, 2013.

[43] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," *IETF, RFC 6330*, 2011.

[44] A. Shokrollahi and M. Luby, "Raptor Codes, Foundations and Trends." Communications and Information Theory, Now Publisher, 2011.

[45] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center," in *Proc. of USENIX*, 2014.

[46] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *In Proc. NSDI, USENIX*, 2015.

[47] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: : A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proc. of SIGCOMM*, 2009.

[48] A. Singla, C.-Y. Hong, L. Popa, and P. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. of USENIX*, 2012.

[49] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," in *ACM SIGCOMM*, 2014.

[50] M. G. Luby, R. Padovani, T. J. Richardson, L. Minder, and P. Aggarwal, "Liquid cloud storage," *CoRR*, vol. abs/1705.07983, 2017. [Online]. Available: http://arxiv.org/abs/1705.07983

[51] P. A. Michael Luby, Lorenz Minder, "Performance of codornicesrq software package," in *International Computer Science Institute*, May, 2019. [Online]. Available: https://www.codornices.info/performance

[52] G. Parisis, G. Xylomenos, and T. Apostolopoulos, "Dhtbd: A reliable block-based storage system for high performance clusters," in *Proc. of IEEE/ACM CCGrid*, 2011.

[53] P. J. Braam, "File systems for clusters from a protocol perspective," http://www.lustre.org.

[54] Z. Guo and Y. Yang, "Multicast fat-tree data center networks with bounded link oversubscription," in *IEEE INFOCOM*, 2013.

[55] M. Alasmar and G. Parisis, "Evaluating modern data centre transport protocols in OMNeT++/INET," in *Proceedings of the 6th OMNeT++ Community Summit Conference*, Hamburg, Germany, 2019.

[56] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. a. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network Albert," in *Proc. of SIGCOMM*, 2009.

Chapter 4

Evaluating Modern Data Centre Transport Protocols in OMNeT++/INET

Preface: paper 3

This chapter includes our paper on developing a simulation framework for evaluating data transport protocols for data centres in OMNeT++ [7]. The paper describes the developed framework, including the fat-tree topology, per-packet and per-flow ECMP, flow scheduling and the NDP model. The paper then evaluates the performance of NDP in OMNeT++ in a simulated fat-tree topology.

- Mohammed Alasmar and George Parisis, "Evaluating Modern Data Centre Transport Protocols in OMNeT++/INET". In Proceedings of the OMNeT++ Community Summit 2019, Hamburg, Germany [7].

Contributions from Co-Authors

The work presented in this paper is substantially my own. I was encouraged by my supervisor to build an OMNeT++ model to simulate data centre protocols, which led to this publication. All the source code that was used in the context of this paper is my own work[1]. My supervisor (and co-author in this paper) provided constructive feedback on the manuscript and the developed framework which contributed to the completion of this work and the respective paper.

[1]https://github.com/mohammedalasmar/ndpTcpDatacentreOmnetppModel

Evaluating Modern Data Centre Transport Protocols in OMNeT++/INET

Mohammed Alasmar and George Parisis

School of Engineering and Informatics
University of Sussex, UK
M.Alasmar@sussex.ac.uk, G.Parisis@sussex.ac.uk

Abstract

In this paper we present our work towards an evaluation platform for data centre transport protocols. We developed a simulation model for NDP[1], a modern data transport protocol in data centres, a FatTree network topology and per-packet ECMP load balancing. We also developed a data centre environment that can be used to evaluate and compare data transport protocols, usch as NDP and TCP. We describe how we integrated our model with the INET Framework and present example simulations to showcase the workings of the developed framework. For that, we ran a comprehensive set of experiments and studied different components and parameters of the developed models.

1 Introduction

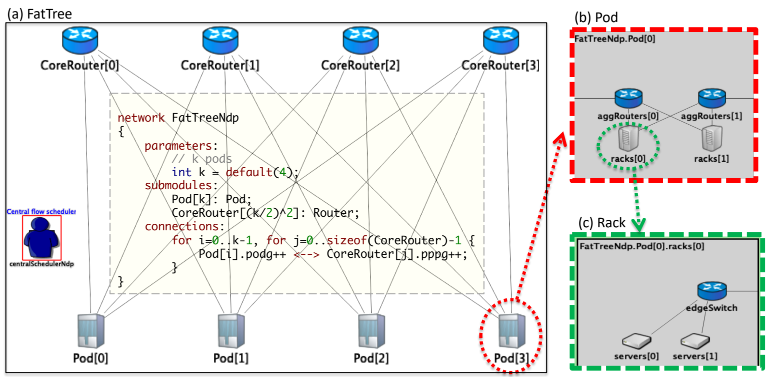
The study of network protocols for Data Centre Networks (DCNs) has become increasingly important, given that data centres support all major Internet services, such as search (e.g. Google), social networking (e.g. Facebook), cloud services (e.g. Amazon EC2) and video streaming (e.g. NetFlix). DCNs consist of a large number of commodity servers and switches and support multiple paths among servers. Recent research on data centre networking is based on various simulation tools and respective models for network protocols [15, 19, 18].
OMNeT++ [25] is an excellent candidate for developing models for data centre networks and respective protocols, and more work is required for establishing it as the de facto simulator for this research community. This is possible through the INET Framework, which is built on top of the simulation core provided by OMNeT++. Omnet++ and INET is built around the concept of modules that communicate by message passing. Protocols are represented by components, which can be combined to form hosts, routers, switches and other networking devices. What makes this framework ideal for evaluating DCN protocols is that new modules can be easily integrated with the existing modules. DCN topologies (e.g. FatTree [1]) can be easily built and parameterised using Omnet++ NED language.
Modelling DCN protocols in OMNeT++. Recently, some DCN-related research has been based on OMNeT++/INET [20, 4, 10, 3]. Achieving the critical mass of researchers that use OMNeT++ for evaluating data centre networks and protocols requires making more modern protocols available in the OMNeT++ environment. Large-scale simulations are crucial for the DCN research community given that access to real-world deployments is very difficult. Developing models for DCNs in OMNeT++ would also ensure reproducibility, revisability (dynamic

[1]https://github.com/mohammedalasmar/ndpTcpDatacentreOmnetppModel (OMNeT++-5.2.1 & INET-3.6.3)

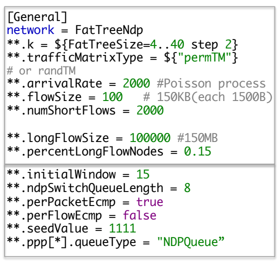
debugging and profiling) and control over the studied traffic workloads (generating realistic traffic workloads in a deterministic fashion) [24].
Efficient data centre transport protocols. In DCNs, an efficient data transport mechanism is crucial to provide near-optimal completion times for short transfers and high goodput for long flows. The performance of TCP in DCNs is problematic due to TCP Incast [9], queue build-up and buffer pressure [13, 23, 16] and per-flow ECMP collisions. TCP performance can get singificantly degraded because of frequent retransmissions of lost packets [5, 16]. Recently, a large body of work aimed at tackling various aspects of data centre transport: proposed approaches usually focus on either achieving low latency [5, 21, 26, 20, 6] or high throughput [16, 22, 11, 2]. NDP [13] appears to perform well with respect to both low latency and high throughput requirements by combining a number of data transport mechanisms.
NDP and FatTree models. In this paper we present an OMNeT++/INET framework for evaluating data transport protocols (NDP and TCP) in data centres. This includes: (1) a model for building FatTree topologies to evaluate the performance of TCP, NDP and other community-developed protocol models for data centres (§2), (2) a model for per-packet and per-flow Equal-Cost Multi-Path (ECMP) load balancing in a FatTree topology (§3), (3) a model for NDP (§4 and §5), and (4) a central traffic scheduler for scheduling flows in the simulated network and setting up simulation parameters for experimenting with the above-mentioned contributions (§5).

2 FatTree Topology

Among the recently proposed DCN topologies, FatTree, which originated from the Clos switching network, is widely used [1]. We developed a FatTree topology generator using the NED language. FatTree data centres allow any two servers to communicate by fully utilising network resources and ensuring non-blocking behaviour. The role of core switches is to forward traffic among aggregation switches, and that of the aggregation switch is to inter-connect core and edge switches. The edge switches reside at lowest level of the topology, and forward traffic between hosts and aggregation switches (see Figures 1a&2). The size of a FatTree topology depends on the number of pods it consists of (k). A FatTree network consists of three layers: the core layer, aggregation layer and edge layer. In a k-ary FatTree topology there are $k^2/4$ core switches which is the same number as the shortest-paths between any two servers that are connected to any pod in the network. Each pod contains k servers and k switches. These switches are divided into two layers each consisting of $k/2$ switches. The first layer is the edge where each switch is connected to $k/2$ of the servers (a rack) in the same pod, while the second layer is the aggregation layer where each switch is connected to $k/2$ of the core switches. Each core switch is connected to one aggregation switch of each pod. The maximum number of servers in a FatTree with k pods is $k^3/4$. All switches have the same number of ports which is equal to k. Table 1 summarises the construction of a k-ary FatTree (with examples when $k =$ 4, 8 and 10).
Generating FatTree networks. The implementation of the FatTree topology using the NED language is based on the values in Table 1. The only required input value to generate the topology is k. Figure 1 shows an example of a generated FatTree topology when $k = 4$. The FatTree module is a network (complex) module that contains two simple modules: Pod and



```
network FatTreeNdp
{
    parameters:
        // k pods
        int k = default(4);
    submodules:
        Pod[k]: Pod;
        CoreRouter[(k/2)^2]: Router;
    connections:
        for i=0..k-1, for j=0..sizeof(CoreRouter)-1 {
            Pod[i].podg++ <--> CoreRouter[j].pppg++;
        }
}
```



```
[General]
network = FatTreeNdp
**.k = ${FatTreeSize=4..40 step 2}
**.trafficMatrixType = ${"permTM"}
# or randTM
**.arrivalRate = 2000 #Poisson process
**.flowSize = 100   # 150KB(each 1500B)
**.numShortFlows = 2000

**.longFlowSize = 100000 #150MB
**.percentLongFlowNodes = 0.15
**.initialWindow = 15
**.ndpSwitchQueueLength = 8
**.perPacketEcmp = true
**.perFlowEcmp = false
**.seedValue = 1111
**.ppp[*].queueType = "NDPQueue”
```

(a) FatTree topology (k=4)

(b) centralScheduler ini file

Figure 1: FatTree implementation in NED language including a central scheduler node

Rack submodules. The NED code for the FatTree module is depicted in Figure 1a, which is used to create a single link between each core switch and each pod.

Pods	k	4	8	10
Servers	$k \times k/2 \times k/2$	16	128	250
Core switches (= servers in each Pod)	$k/2 \times k/2$	4	16	25
Edge switches in each Pod (racks)	$k/2$	2	4	5
Aggregation switches in each Pod	$k/2$	2	4	5
Total edge/aggregation switches	$k \times k/2$	8	24	50
Switch ports	k	4	8	10
Equal-cost path between any pair of servers (at different pod)	$k/2 \times k/2$	4	16	50

Table 1: k-ary FatTree topology architecture: examples when k = 4, 8 and 10

3 Per-packet and per-flow ECMP

Modern data centre transport protocols exploit the existence of multiple equal-cost paths in FatTree networks to better balance traffic in the network, eliminate hotspots and achieve high throughput. ECMP is used for packet forwarding in the network [14]. **In per-flow ECMP**, packets are classified into different flows by hashing the each packet's 5-tuple (source IP address, destination IP address, protocol number, source port number and destination port number). Packets of the same flow go over the same link, as depicted in Figure 2a. Per-flow hashing ensures that packets belonging to the same flow (or sub-flow in MultiPath TCP) will arrive in order to their destination. However, this can cause significant underutilisation in the network due to collisions of large flows (i.e. when a large number of flows cross the same link while other links are not used) and this can significantly reduce throughput, as discussed in [12]. In addition, per-flow load-balancing can result in unequal link utilisation and hotspots. **In**

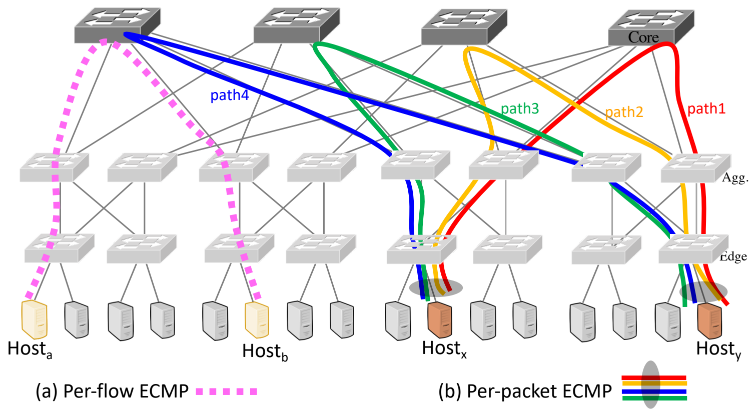


Figure 2: (a) Per-flow ECMP vs (b) per-packet ECMP in 4-FatTree

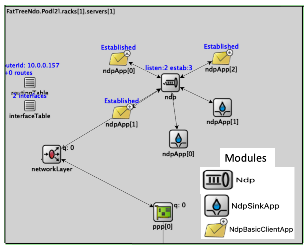


Figure 3: NDP modules

per-packet ECMP, packet forwarding is randomised over all equal-cost links used for load balancing, as shown in Figure 2b. Per-packet multipath forwarding is a good option when using a data transport protocol that can tolerate reordering (e.g., NDP [13][3]). As per-packet ECMP may result in packets arriving out of order, it cannot be used with data transport protocols that are sensitive to packet reordering (e.g., TCP).

A model of per-packet and per-flow ECMP in INET. We implemented both per-packet and per-flow ECMP by updating the source code that is provided in INETv3.6.3 network layer. Routing in INET is done in five main steps as follows: (1) building topology and assigning addresses (*NetworkConfiguratorBase::Topology*), (2) setting links and node weights (*IPv4NetworkConfigurator::computeConfiguration*), (3) using Dijkstra's Algorithm for multiple paths (*Topology::calculateWeightedSingleShortestPathsTo*), (4) adding a route to all destinations in the network (*IPv4NetworkConfigurator::addStaticRoutes*) and (5) generating the routing tables (*IPv4RoutingTable::printRoutingTable*).

When implementing ECMP, we updated step 3 so that all shortest paths to all destinations are registered. Additionally, we updated step 4 to include the updates in step 3 when adding routes to all destinations. The routing tables in step 5 are automatically updated. Finally, we implemented the hashing function in *IPv4RoutingTable::findBestMatchingRouteEcmp* (which is called by *IPv4::routeUnicastPacket*). There are two options for hashing:

- per-packet ECMP: *selectPath = rand()% numPossibleEcmpRouts*, and
- per-flow ECMP: *selectPath = hashValue% numPossibleEcmpRouts*, where the *hashValue* is calculated based on the 5-tuple (we also included the router's name in this hashing).

4 A model of NDP in INET

NDP aims at offering both low latency and high throughput in FatTree data centre networks. NDP combines several ideas into a clean slate protocol design. NDP exhibits very good performance by employing receiver-driven flow control and packet trimming. NDP will be deployable when P4 switches [8] are deployed in data centres. NDP operation can be summarised as follows (and depicted through the numbered circles in Figure 4). Senders are allowed to send an initial window of data at line rate (circles 1 and 2). Switches use shallow buffers (e.g. 8 packets long) with two queues: the data (used for data packets only) and control (high priority) queue (for PULL, ACK, NACK and Header packets). If the data queue overflows, the packet payload is trimmed and the header is priority-forwarded (circles 3, 4, 5 and 6). At the receiver, an ACK for each data packet received and a NACK for each header will be sent immediately to the sender (circles 7 and 8). The receiver has a shared PULL queue between all active connections. The receiver adds a PULL packet for every received header or data packet (circle 9). The receiver paces PULL packets so as to fill the receiver's incoming link. Pacing is across all connections, so that the aggregate rate matches the receiver's link speed (circle 10). The goal is to keep the incoming link full, so the receiver spaces pull requests accordingly (e.g. assuming each incoming packet has the same MTU size = 1500B, then the receiver sends a packet every $\frac{MTU}{1Gbps} = 12\mu$ seconds if the receiver's link speed is 1Gbps). At the sender, PULL requests trigger either a retransmission or a new data packet (circle 11).

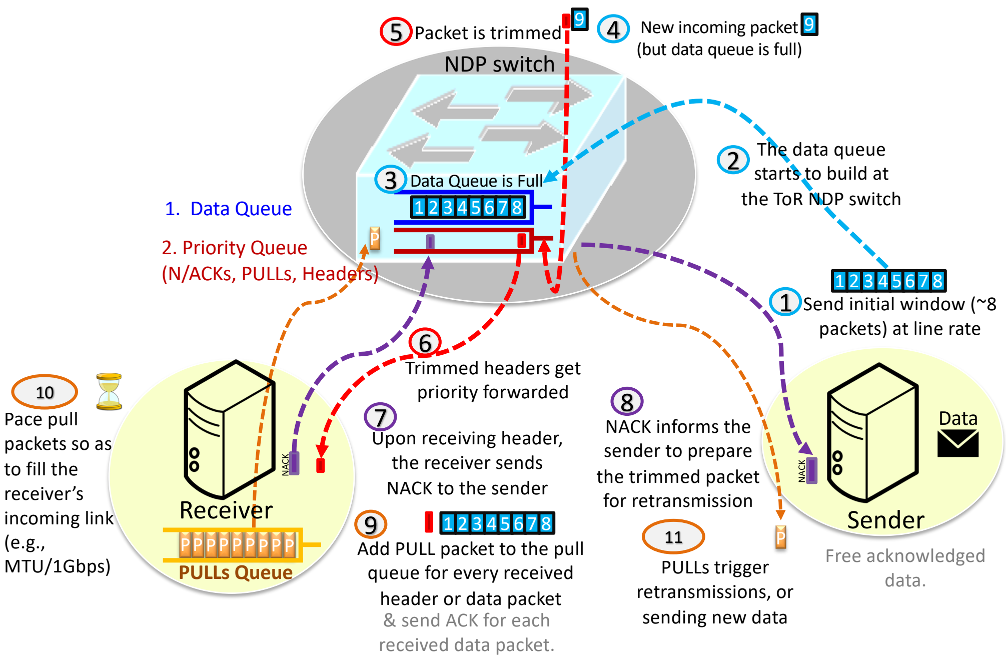

5 Packet is trimmed
9
4 New incoming packet 9
(but data queue is full)
NDP switch
2 The data queue starts to build at the ToR NDP switch
3 Data Queue is Full
12345678
1. Data Queue
2. Priority Queue (N/ACKs, PULLs, Headers)
12345678
1 Send initial window (~8 packets) at line rate
6 Trimmed headers get priority forwarded
10 Pace pull packets so as to fill the receiver's incoming link (e.g., MTU/1Gbps)
Receiver
PPPPPPPPP
PULLs Queue
7 Upon receiving header, the receiver sends NACK to the sender
8 NACK informs the sender to prepare the trimmed packet for retransmission
Data
Sender
Free acknowledged data.
9 12345678
Add PULL packet to the pull queue for every received header or data packet
& send ACK for each received data packet.
11 PULLs trigger retransmissions, or sending new data
NACK
P

Figure 4: NDP operation [2]

Here we describe how we implemented NDP in INET. Our implementation follows the TCP model in INET. We developed *StandardHostNdp*, a predefined NED type which is an OMNeT++ compound module that is composed of the following components:

Applications. There are two main applications that can communicate with the NDP layer, as shown in Figure 3. The first application is the *NdpBasicClientApp* module, which is used

[2]This is an abstract diagram of NDP functionality – the details of NDP can be found in [13]

by NDP senders to start a connection. The second application module is *NdpSinkApp* which is used by NDP receivers to listen for incoming connections. NDP applications and the NDP layer communicate with each other by sending *cMessage* objects. These messages are specified in the *NDPCommand.msg* file. The *NDPCommandCode* enumeration defines the types of messages that are sent by the application to the NDP layer. These are the main message types: *NDP_C_OPEN_ACTIVE*: active open, *NDP_C_OPEN_PASSIVE*: passive open, *NDP_C_SEND*: send data and *NDP_C_CLOSE*: no more data to send. Each command message should have attached control information of type *NDPCommand*. For example, the command message *NDP_C_OPEN_ACTIVE* requires this control information to be attached: *connId*, *localAddr*, *locarPrt*, *remoteAddr*, *remotePrt* and *numPacketsToSend*. In Figure 5, *NdpSinkApp* can open a local port by sending a *NDP_C_PASSIVE_OPEN* to the NDP layer with control information that contains the local address and port. *NdpBasicClientApp* opens a connection to a remote server by sending a *NDP_C_OPEN_ACTIVE* command to the NDP layer.

NDP layer. The NDP module creates an *NDPConnection* object upon receiving either an active or passive open command from an NDP application. The main message kinds that the application receives from the NDP layer are: *NDP_I_ESTABLISHED*: connection established, *NDP_I_DATA*: data packet received and *NDP_I_PEER_CLOSED*: FIN flag received from remote NDP. The NDP module and the NDP applications implement NDP operations (as discussed in §4) as follows:

1. The NDP sender performs the following operations (see Figure 5):
   - creates the sendQueue for the data to be sent: *NDPMSgBasedSendQueue::init(data)*,
   - sends the initial window of data packets with SYN flag: *sendInitialWindow()*,
   - precesses received packet (can be ACK, NACK or PULL) from the receiver: *NDPConnection::process_RCV_Pkt(ndpPkt)*. The sender takes one of the following actions:
     - frees acknowledged data packet: *sendQueue->ackArrivedFree(ndpPkt)*,
     - queues NACKed data packet for retransmission: *nackArrivedMoveFront(ndpPkt)*,
     - sends data packet if PULL request arrived: *sendQueue->sendNdpPkt()*,
   - sets the FIN flag when sending the last data packet: *ndpPkt->setFinFlag(true)*.

2. The NDP receiver performs the following operations (see Figure 5):
   - establishes a connection when receiving a data packet with the SYN flag set: *NDPConnection:processPktInListen(pktSynTrue)*,
   - acknowledges a received data packet: *sendAckPkt(seqNo)*,
   - sends NACK when receiving header packet: *sendAckPkt(seqNo)*,
   - adds PULL packet to the PULL queue upon receiving either data or header packet: *addRequestToPullQueue()*,
   - schedules a self message that is used to pace pull packets from the PULL queue: *ScheduleAt(PACING_TIME, pullTimerMsg)*
   - removes all pull packets when receiving the last data packet: *pullQueue->removePulls()*

Network layer. The IPv4 module performs IP encapsulation/decapsulation and routing of datagrams. This is based on function call interface of the IPv4RoutingTable which we updated as discussed in §3.

NICs. We use the available NIC modules (*PPPInterface* and *EthernetInterface*) that are provided in the INET Framework.
NDP Switch. It contains two queues: (1) *highPriorityQueue*: which is used to enqueue Header, ACK, NACK and PULL packets, and (2) *dataQueue*: which is used to enqueue data packets. NDP switches trim packets when their *dataQueue* is full and insert the headers in the *highPriorityQueue*. *NDPSwitch:dequeue()* is round robin (10:1 ratio) between the two queues.

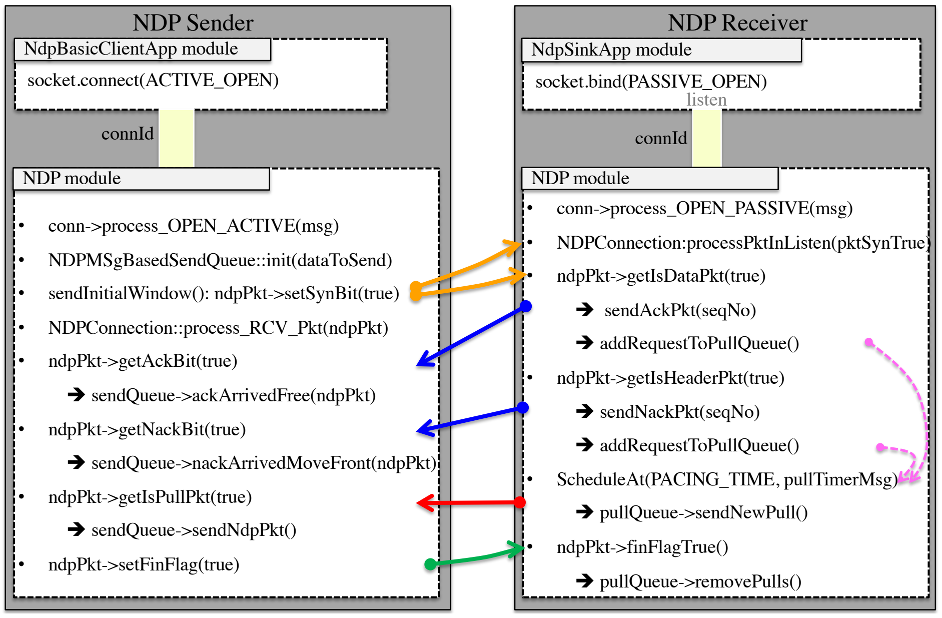

NDP Sender
NdpBasicClientApp module
socket.connect(ACTIVE_OPEN)
connId
NDP module
conn->process_OPEN_ACTIVE(msg)
NDPMSgBasedSendQueue::init(dataToSend)
sendInitialWindow(): ndpPkt->setSynBit(true)
NDPConnection::process_RCV_Pkt(ndpPkt)
ndpPkt->getAckBit(true)
sendQueue->ackArrivedFree(ndpPkt)
ndpPkt->getNackBit(true)
sendQueue->nackArrivedMoveFront(ndpPkt)
ndpPkt->getIsPullPkt(true)
sendQueue->sendNdpPkt()
ndpPkt->setFinFlag(true)
NDP Receiver
NdpSinkApp module
socket.bind(PASSIVE_OPEN)
listen
connId
NDP module
conn->process_OPEN_PASSIVE(msg)
NDPConnection:processPktInListen(pktSynTrue)
ndpPkt->getIsDataPkt(true)
sendAckPkt(seqNo)
addRequestToPullQueue()
ndpPkt->getIsHeaderPkt(true)
sendNackPkt(seqNo)
addRequestToPullQueue()
ScheduleAt(PACING_TIME, pullTimerMsg)
pullQueue->sendNewPull()
ndpPkt->finFlagTrue()
pullQueue->removePulls()

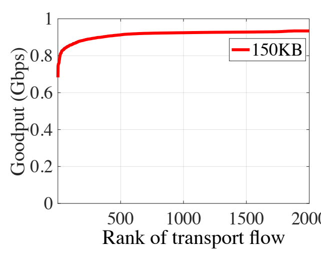
Figure 5: NDP implementation

5 Experimenting with the Simulation Framework

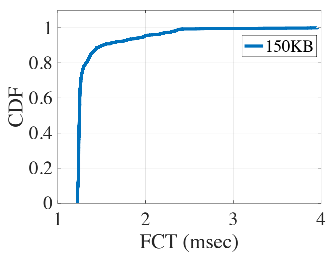
We developed a central scheduler node (*centralSchedulerNdp*[3]) to schedule NDP connections in a FatTree topology (as shown in Figure 1). The main parameters in this node are shown in Figure 1b and explained below:

- *FatTreeSize*. This is used to generate a k-ary FatTree topology (as described in §2)
- *trafficMatrixType*. The scheduler node can schedule either a permutation or random traffic matrix (as explained in[17])
- *arrivalRate*. This is used to generate flow arrival times according to a Poisson process. This is implemented by creating an exponential distribution (mean = $1/\lambda$) based on the Poisson arrival rate λ.
- *flowSize*. This is the size of the data (number of packets) that each created *NdpBasicClientApp* is going to send to an *NdpSinkApp*.

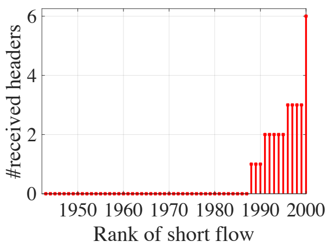
[3]We have developed another scheduler: *centralSchedulerTcp*, which is used for TCP experiments and it has a similar interface as the NDP scheduler. It is used to schedule TCP connections in a FatTree topology.



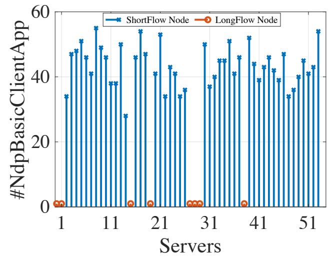
(a) Goodput



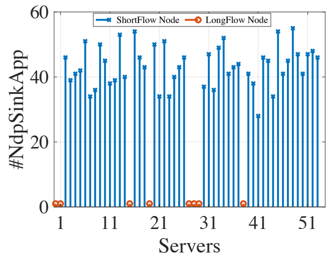
(b) FCT



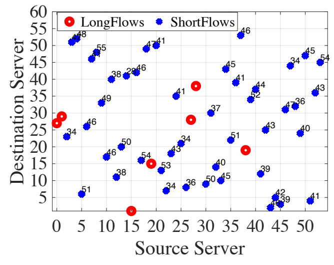
(c) Num. of received headers



(d) #created sender's app.



(e) #created receiver's app.



(f) Perm. traffic matrix

Figure 6: The results of running 2000 NDP short flows (150KB each) in a 6-FatTree (54 servers)

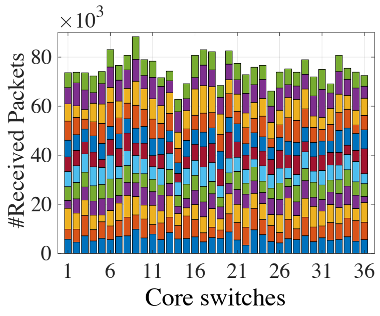
- *numShortFlows*. This is the number of NDP connections that will be scheduled based on the selected traffic matrix, e.g. scheduling 1000 flows means that there are 1000 different *NdpBasicClientApps* communicate with 1000 *NdpSinkApps*.
- *longFlowSize* and *percentLongFlowNodes*. These parameters are used to generate background traffic in the simulation. The background traffic can either run until all short flows will finish or by assigning the size of the long flows to a large flow size. For example, *percentLongFlowNodes* of 30% in a 10-FatTree (250 servers) topology means that there are 75 servers that are used only to run background traffic.
- *initialWindow* and *ndpSwitchQueueLength*. These are used as described in Section refndp-section).

5.1 NDP Experiments

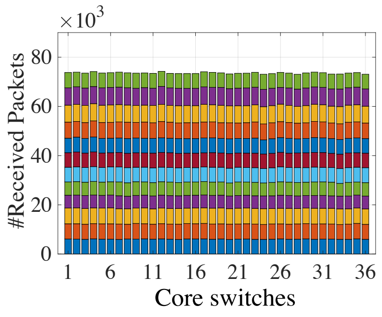
Here, we present experimentation using the developed NDP model. We conducted four different experiments[4].

Experiment 1. We ran this experiment with the parameters that are shown in Figure 1b in a 6-FatTree (54 servers) where 15% of servers (8 servers) ran background load. The number of short flows were 2000 and each flow had a size of 150KB. The results of this experiment are shown in Figure 6 (one seed was used). The developed model provides the ability to record and plot the following measurements: goodput (Figure 6a), flow completion time (FCT) (Figure 6b), the number of received header packets per flow (connection) (Figure 6c), the number of data sending

[4]Due to space limitations we will not discuss simulations with TCP. Some of these results can be found in [3].



(a) Per-flow ECMP



(b) Per-packet ECMP

Figure 7: Per-flow vs per-packet ECMP in 12-FatTree (36 core switches) topology. Each colour in each bar (12 colours) represents the number of received packets at each port (12-port switch)

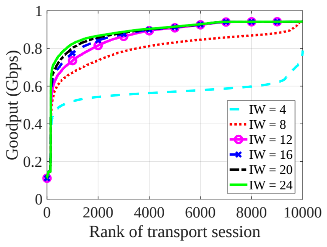
applications at each server (Figure 6d), the number of data receiving applications at each server (Figure 6d) and traffic matrix plot (Figure 6e). We use OMNeT++'s *scavetool* command-line tool to process the results. These scripts can be found in the shell file (*runNdpSimulation.sh*) that we use to run the simulation and generate several *.csv* files which are input to MATLAB scripts to produce the required figures [5]. The results are as expected; most flows achievedvery high goodput and low Flow Completion Times.

Experiment 2. In this experiment, we compare between per-flow and per-packet ECMP in a FatTree topology. We have simulated the two protocols in a 12-FatTree (36 core switches) by running 5000 short flows (300KB each) and with arrival rate $\lambda = 2000$ (without introducing any background load). The other parameters are as depicted in Figure 1b. Figure 7 illustrates the number of received packets at each port of the 36 12-port core switches. It is obvious that per-packet ECMP provides better load balancing than per-flow ECMP (as discussed in §3).

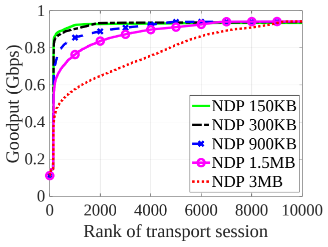
Experiment 3. In this experiment we test the performance of NDP when varying specific parameters. A key parameter of NDP is the initial window of packets that a sender pushes to the network. In Figure 8a, we observe that for very small values of the initial window, the goodput is very low and the receiver's downlink underutilised. As the window increases, utilisation approaches the maximum available link capacity (for 16 packets). For larger values, the measured goodput is the same (full link capacity). In Figure 8b, we show the goodput of NDP flows at different flow sizes (when $\lambda = 2500$). Similarly, in Figure 8b, we obtain the goodput of 10000 flows (1.5MB each) at different λ values.

Experiment 4. In the experiment, we use realistic web search workloads reported by data centre operators to evaluate NDP [5]. We simulate two target loads of background traffic (0.5 and 0.8). We generate 20000 flows, with $\lambda = 2500$. We report the flow completion time (FCT) of all flows (split in two different ranges), as shown in Figure 9. NDP achieves low FCT.

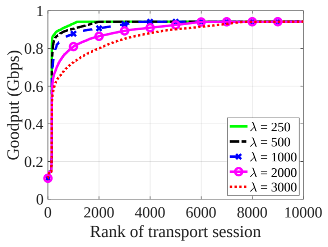
[5]The shell and MATLAB scripts are available in the GitHub repository, along with a demo.



(a) Varying initial window

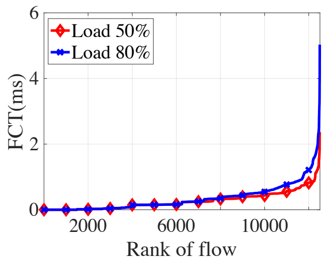


(b) Varying flow size

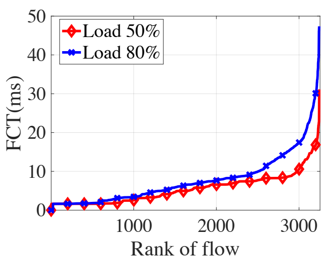


(c) Varying arrival rate λ

Figure 8: NDP Goodput for 10000 flows in a 10-FatTree when varying specific parameter



(a) Flow size (0, 100KB]



(b) Flow size (100KB, 1MB]

Figure 9: FCT of web search workload flows (two ranges) at different background loads

6 Summary and future work

The proposed model is intended to be used to evaluate NDP and compare its performance with existing models (e.g. TCP) in a FatTree data centre topology. The current implementation supports most of NDP's specifications. The current version of our NDP model does not support priority pulling at the pull queue. As a future work, we plan to improve the NDP model by allowing priority pacing in the pull queue for short flows. In addition, we aim at leveraging multiple priority queues available in commodity switches to implement a Multiple Level Feedback Queue (MLFQ). Also, we aim at evaluating NDP in Incast and Outcast scenarios and assess the provided fairness among competing flows. Furthermore, we plan to build more OMNeT++/INET-based models that simulate other modern data transport protocols for DCNs, such as DCTCP[5], MPTCP[23], DCQCN[27] and PIAS[7].

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proc. of SIGCOMM*, 2008.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. of USENIX*, 2010.

[3] Mohammed Alasmar, George Parisis, and Jon Crowcroft. Polyraptor: Embracing path and data redundancy in data centres for efficient data transport. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018.

[4] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman. Data center transport mechanisms: Congestion control theory and ieee standardization. In *Annual Allerton Conference on Communication, Control, and Computing*, 2008.

[5] M Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[6] Mohammad Alizadeh, S Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proc. of SIGCOMM*, 2013.

[7] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-agnostic flow scheduling for commodity data centers. In *In Proc. NSDI, USENIX*, 2015.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. In *ACM SIGCOMM*, 2014.

[9] Yanpei Chen, Rean Griffith, David Zats, Anthony D. Joseph, and Randy Katz. Understanding TCP incast and its implications for big data workloads. In *Proc. of USENIX*, 2012.

[10] T. Das and K. M. Sivalingam. Tcp improvements for data center networks. In *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, 2013.

[11] A Dixit, P Prakash, Y C Hu, and R R Kompella. On the impact of packet spraying in data center networks. In *Proc. of INFOCOM*, 2013.

[12] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *SIGCOMM*, 2016.

[13] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. of SIGCOMM*, 2017.

[14] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm Status. *IETF, RFC 2992*, 2000.

[15] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 2010.

[16] M. Kheirkhah, I. Wakeman, and G. Parisis. MMPTCP: A multipath transport protocol for data centers. In *Proc. of INFOCOM 2016*, 2016.

[17] Morteza Kheirkhah. Mmptcp: a novel transport protocol for data centre networks[. In *University of Sussex*, 2016.

[18] Morteza Kheirkhah, Ian Wakeman, and George Parisis. Multipath-TCP in ns-3. In *WNS3*, 2014.

[19] A. R. A. Kumar, S. V. Rao, and D. Goswami. Ns3 simulator for a study of data center networks. In *IEEE International Symposium on Parallel and Distributed Computing*, 2013.

[20] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *In Proc. ACM SIGCOMM*, 2018.

[21] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *IEEE INFOCOM*, 2013.

[22] Costin Raiciu, Sebastien Barre, Chris Pluntke, Adam Green, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, 2011.

[23] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. Data Center Networking with Multipath TCP. In *Proc. of SIGCOMM*. ACM,

2010.

[24] Bilel Ben Romdhanne. Large-scale network simulation over heterogeneous computing architecture. In *Telecom ParisTech*, 2013.

[25] A. Varga. Omnet++ discrete event simulation. In *System User Manual*, 2006.

[26] H. Xu and B. Li. Repflow: Minimizing flow completion times with replicated flows in data centers. In *IEEE INFOCOM*, 2014.

[27] Y Zhu, H Eran, Dan F, Chuanxiong G, Marina L, Ye Liron, J Padhye, Shachar R, Mohamad Haj Y, and Ming Zhang. Congestion control for large-scale rdma deployments. In *SIGCOMM*, 2015.

Chapter 5

On the Distribution of Traffic Volumes in the Internet and its Implications

Preface: paper 4

This chapter includes our paper on the characterisation of Internet traffic volume using a robust statistical approach [8]. The findings suggest that the log-normal model is a good distribution to fit Internet traffic volumes. The work has significant utility for network operators for network dimensioning and traffic billing purposes.

- Mohammed Alasmar, George Parisis, Richard Clegg and Nickolay Zakhleniuk, "On the Distribution of Traffic Volumes in the Internet and its Implications". In Proceedings of IEEE INFOCOM 2019, Paris, France [8].

Contributions from Co-Authors

The research presented in this paper is substantially my own. The idea of studying the characteristics of Internet traffic has been initially suggested by Nickolay Zakhleniuk, who contributed to this paper by highlighting the limitations of some current approaches that study Internet traffic volumes. Nickolay also helped in improving the quality of the paper by his general comments on the text and suggestions on adding more tests to evaluate our approach. The work was developed to its current form as part of an effort to understand Internet traffic and design network protocols, at the first stages of my PhD. The statistical approach [44] that I used in this paper was suggested by Richard Clegg, who also provided valuable feedback to improve the paper through several online discussions. Richard and my supervisor contributed in shaping the research question, presenting the motivation and conducting the experimental evaluation. All co-authors contributed in improving the published manuscript.

On the Distribution of Traffic Volumes in the Internet and its Implications

Mohammed Alasmar
Department of Informatics
University of Sussex
Brighton, UK
m.alasmar@sussex.ac.uk

George Parisis
Department of Informatics
University of Sussex
Brighton, UK
g.parisis@sussex.ac.uk

Richard Clegg
School of Computer Science
Queen Mary University of London
London, UK
r.clegg@qmul.ac.uk

Nickolay Zakhleniuk
School of Computer Science
University of Essex
Colchester, UK
naz@essex.ac.uk

***Abstract*—Getting good statistical models of traffic on network links is a well-known, often-studied problem. A lot of attention has been given to correlation patterns and flow duration. The distribution of the amount of traffic per unit time is an equally important but less studied problem. We study a large number of traffic traces from many different networks including academic, commercial and residential networks using state-of-the-art statistical techniques. We show that the log-normal distribution is a better fit than the Gaussian distribution commonly claimed in the literature. We also investigate a second heavy-tailed distribution (the Weibull) and show that its performance is better than Gaussian but worse than log-normal. We examine anomalous traces which are a poor fit for all distributions tried and show that this is often due to traffic outages or links that hit maximum capacity.**

We demonstrate the utility of the log-normal distribution in two contexts: predicting the proportion of time traffic will exceed a given level (for service level agreement or link capacity estimation) and predicting 95th percentile pricing. We also show the log-normal distribution is a better predictor than Gaussian or Weibull distributions.

***Index Terms*—Traffic modelling, network planning, bandwidth provisioning, traffic billing**

I. INTRODUCTION

Internet traffic characterisation is an important problem for network researchers and vendors. The subject has a long history. Early works [1], [2] discovered that the correlation structure of traffic exhibits self-similarity and that the durations of individual flows of packets exhibit heavy-tails [3]. These works were later challenged and refined (see Section VI for a summary). By comparison the distribution of the amount of traffic seen on a link in a given time period has seen comparatively less research interest. This is surprising as this quantity can be extremely useful in network planning.

In this paper we use a rigorous statistical approach to fitting a statistical distribution to the amount of traffic within a given time period. Formally, we choose some timescale T and let X_i be the amount of traffic seen in the time period $[iT, (i+1)T)$. We investigate the distribution of the random variable X over a wide range of values of T. We show that the distribution of the variable has considerable implications for network planning; for assessing how often a link is over capacity and in particular for service level agreements (SLAs), and for traffic pricing, particularly using the 95th percentile scheme [4].

Previous authors have claimed that X has a normal (or Gaussian) distribution [5]–[7]. Others claim X is Gaussian plus a tail associated with bursts [8], [9]. A variable X has a log-normal distribution if its logarithm is normally distributed $\ln(X) \sim N(\mu, \sigma^2)$ where $\mu \in \mathbb{R}$ is the mean and $\sigma > 0$ is the standard deviation of the distribution. We use a well-established statistical methodology [10] to show that a log-normal distribution is a better fit than Gaussian or Weibull[1] for the vast majority of traces. This holds over a wide range of timescales T (from 5 msec to 5 sec). This paper is the most comprehensive investigation of this phenomenon the authors know about. We study a large number of publicly available traces from a diverse set of locations (including commercial, academic and residential networks) with different link speeds and spanning the last 15 years.

The structure of the paper is as follows. In Section II we describe the datasets used. In Section III we describe our best-practice procedure for fitting traffic and demonstrate that log-normal is the best fit distribution for our traces under a variety of circumstances. We examine those few traces that do not follow this distribution and find it occurs when a link spends considerable time either having an outage or completely at maximum capacity. In Section IV we demonstrate that the log-normal distribution is the most useful for estimating how often a link is over capacity. In Section V we show that the log-normal distribution provides good estimates when looking at 95th percentile pricing. In Section VI we give related work. Finally, Section VII gives our conclusions.

II. NETWORK TRAFFIC TRACES

A key contribution of our work stems from the spatial and temporal diversity of the studied traces. The dataset spans a period of 15 years and comprises 229 traces.

CAIDA traces. We have used 27 CAIDA traces captured at an Internet data collection monitor which is located at an Equinix data centre in Chicago [11]. The data centre is connected to a backbone link of a Tier 1 ISP. The monitor records an hour-long traces four times a year, usually from 13:00 to 14:00 UTC. Each trace contains billions of IPv4 packets, the headers of which are anonymised.

[1]A variable X has a Weibull distribution with parameters $k > 0$ (known as shape) and $\lambda > 0$ (known as scale) if its probability density function follows $f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} \exp(-(x/\lambda)^k)$ when $x \geq 0$ and is 0 otherwise.

The average captured data rate is 2.5 Gbps. At the time of capturing, the monitored link had a capacity of 10 Gbps. Traces were captured between 2013 and 2016.
MAWI traces. The MAWI archive [12] consists of a collection of Internet traffic traces, captured within the WIDE backbone network that connects Japanese universities and research institutions to the Internet. Each trace consists of IP level traffic observed daily from 14:00 to 14:15 at a vantage point within WIDE. Traces include anonymised IP and MAC headers, along with an *ntpd* timestamp [12]. We have looked at 107 traces (each one being 15 minutes long). Traces were captured between 2014 and 2018. On average, each trace consists of 70 million packets; the average captured data rate is 422 Mbps. The monitored link had a capacity of 1 Gbps.
Twente University traces. We used 40 traffic traces captured at five different locations (8 traces from each location). Traces are diverse in terms of the link rates, types of users and captured time [13]. Each trace is 15 minutes long. The first location is a residential network with a 300 Mbps link, which connects 2000 students (each one having a 100 Mbps access link); traces were captured in July 2002. The second location is a research institute network with a 1 Gbps link which connects 200 researchers (each one having a 100 Mbps access link); traces were captured between May and August 2003. The third location is at a large college with a 1 Gbps link which connects 1000 employees (each one having a 100 Mbps access link); traces were captured between February and July 2004. The fourth location is an ADSL access network with a 1 Gbps ADSL link used by hundreds of users (each one having a 256 Kbps to 8 Mbps access link); traces were captured between February and July 2004. The fifth location is an educational organisation with a 100 Mbps link connecting 135 students and employees (each one having a 100 Mbps access link); traces were captured between May and June 2007.
Waikato University VIII traces. The Waikato dataset consists of traffic traces captured by the WAND group at the University of Waikato, New Zealand [14]. The capture point is at the link interconnecting the University with the Internet. All of the traces were captured using software that was specifically developed for the Waikato capture point and a DAG 3 series hardware capture card. All IP addresses within the traces are anonymised. In our study, we have used 30 traces captured between April 2011 and November 2011.
Auckland University IX traces. The Auckland dataset consists of traffic traces captured by the WAND group at the University of Waikato [15]. The traces were collected at the University of Auckland, New Zealand. The capture point is at the link interconnecting the University with the Internet. All IP addresses within the traces are anonymised. In our study, we have used 25 traces captured in 2009.

III. Fitting a statistical distribution to Internet traffic data

In this section we present an extensive statistical analysis applied to the datasets described in the previous section. The aim is to discover which statistical distribution best fits the traces. In contrast to the existing research (see Section VI), we are basing our analysis on the framework proposed by Clauset et al. [10], a comprehensive statistical framework developed specifically for testing power-law behaviour in empirical data[2]. The framework combines maximum-likelihood fitting methods with goodness-of-fit tests based on the Kolmogorov–Smirnov statistic and likelihood ratios. The method reliably tests whether the power-law distribution is the best model for a specific dataset, or, if not, whether an alternative statistical distribution (e.g., log-normal, exponential, Weibull) is. The framework performs the tests described above as follows: (1) the parameters of the power-law model are estimated for a given dataset; (2) the goodness-of-fit between the data and the power-law is calculated, under the hypothesis that the power-law is the best fit to the provided traffic samples. If the resulting p-value is greater than 0.1 the hypothesis is accepted (i.e. the power law is a plausible fit to the given data), otherwise the hypothesis is rejected; (3) alternative distributions are tested against the power-law as a fit to the data by employing a likelihood ratio test.

For the vast majority of the traces examined, the hypothesis was rejected; i.e. the power-law distribution was not a good fit. Consequently, we investigate alternative distributions by performing the likelihood ratio (LLR) test (following Clauset's methodology), as follows:

$$\Re, p = \mathrm{fit.distributionCompare}(\mathrm{powerlaw}, \mathrm{alternative})$$

where $\Re$ is the normalised LLR[3] between the power-law and alternative distributions and p is the significance value for this test. $\Re$ is positive if the power-law distribution is a better fit for the data, and negative if the alternative distribution is a better fit for the data. A p-value less than 0.1 means that the value of $\Re$ can be trusted to make a conclusion that one candidate distribution (power-law or alternative, depending on the sign of $\Re$) is a good fit for the data. In contrast, a p-value greater than 0.1 means that there is nothing to be concluded from the likelihood ratio test.

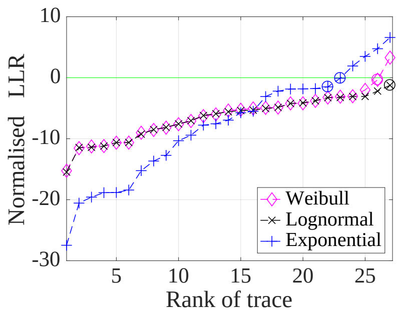
A. Fitting the log-normal distribution to Internet traffic data

Figure 1 shows the results of the LLR test for all 229 traces with log-normal, exponential and Weibull distribution as the alternative to power-law. For this test we have aggregated traffic at a timescale $T = 100$ msec. The points marked with a circle are the ones with $p > 0.1$. It is clear that the log-normal distribution (black line in Figure 1) is the best fit for the studied traces; i.e. $\Re < 0$ and $p < 0.1$ for most traces when the alternative distribution (to the power-law which is almost always rejected) is the log-normal one[4]. The log-normal distribution is not the best fit for 1 out of 27 CAIDA traces, 2 out 30 Waikato traces, 1 out of 25 Auckland traces, 5 out of

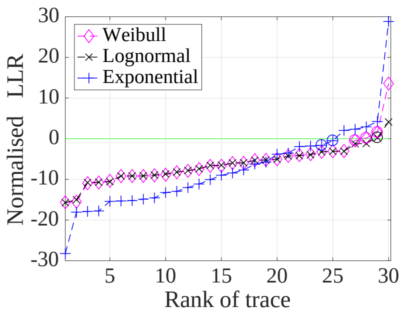
[2]We have used the source code discussed in [16].

[3]$\Re$ is calculated as $\mathcal{R}/(\sigma\sqrt{n})$, where $\mathcal{R}$ is the log likelihood ratio [10].

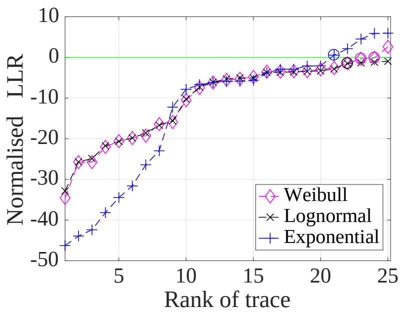
[4]For clarity, in Figures 1(e) and 2(e) we only plot traces 60 – 107. For traces 1 – 59, $\Re$ is less than 0 and the respective p-value is less than 0.1; i.e. the alternative distribution is the best fit for the respective trace



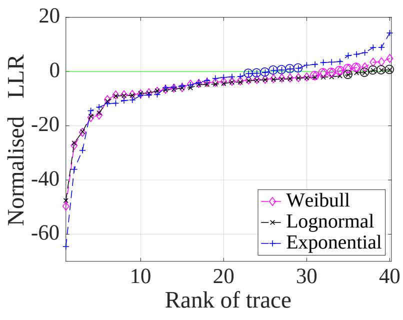
(a) CAIDA traces



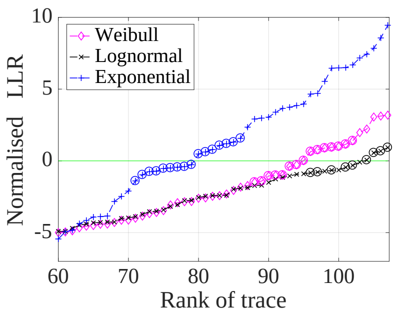
(b) Waikato traces



(c) Auckland traces



(d) Twente traces



(e) MAWI traces

Fig. 1: Normalised Log-Likelihood Ratio (LLR) test results for all studied traces and candidate distributions. Aggregation timescale T is 100 msec. Circled points in the plot are the ones with p-value greater than 0.1; i.e. likelihood test is inconclusive with respect to fitting any of the candidate distributions to the traffic data.

40 Twente traces and 9 out of 107 MAWI traces. We examined these traces in more detail and discuss them in Section III-B.

For the vast majority of traces the power-law distribution is favoured over the exponential one (i.e. $\Re > 0$), as shown in Figure 1. Thus, the exponential distribution cannot be considered as a good model for our traffic traces. On the other hand, the Weibull appears to be a better fit over the power-law distribution; however, when compared to the log-normal distribution, it still performs poorly (i.e. $\Re > 0$ or $\Re < 0$ but $p > 0.1$) for a substantial amount of traces.

Identifying the log-normal distribution as the best fit for the vast majority of traffic traces at $T = 100$ msec is very encouraging. This specific traffic aggregation timescale has been commonly studied in the literature [17], [18]. Next we investigate what the best model is for a range of aggregation timescales. The results are shown in Figure 2. As reflected by the $\Re$ and p-values, the log-normal distribution is the best fit for the vast majority of captured traces at all examined timescales (5 msec to 5 sec)[5]. This is a strong result suggesting the generality of our observations. The good log-normal fit at time scales as small as 5 msec is important for practical applications of the log-normal model.

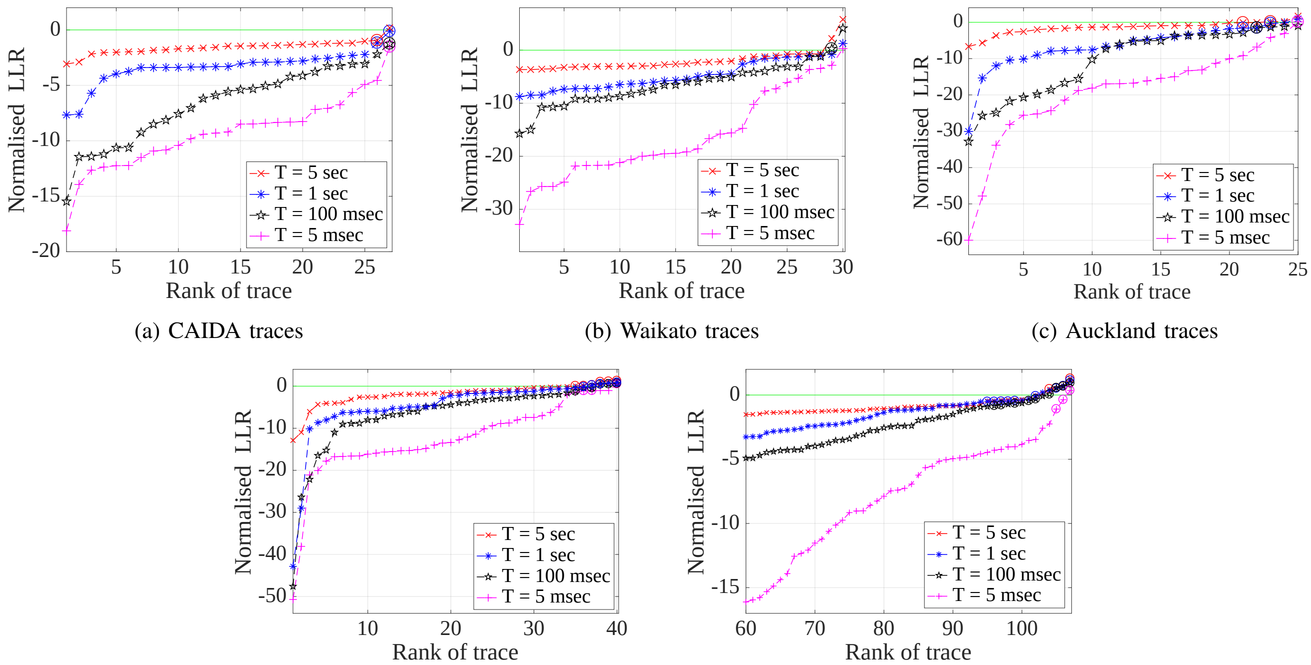
We also examined Q-Q plots for a large number of traces[6]. The log-normal distribution appeared to be a better fit than other tested distributions and no deviations from the expected pattern were observed in the body or tail of the distribution.

B. *Anomalous traces*

As mentioned in Section III-A, there are a small number of traces for which the log-normal distribution is not a good fit (none of the other examined distributions is, either). Figure 3(a) shows the PDF plot for one of the 8 anomalous MAWI traces. Figure 3(b) shows the PDF for another MAWI trace for which the log-normal distribution is a good fit. It is obvious from Figure 3(a) that the link was either severely underutilised (see large spike on the left part of the plot area) or fully utilised (see smaller spike at the right part of the plot area) for higher data rates. All traces for which the log-normal distribution was not a good fit exhibited similar behaviour and (aggregated) traffic patterns. On the contrary, we did not observe any such behaviour for the majority of traces for which the log-normal distribution was the best fit. A likely explanation for the anomalous traces is that those traces contain either periods of over-capacity (traffic is at 100% of link capacity) or periods where the link is broken (no traffic).

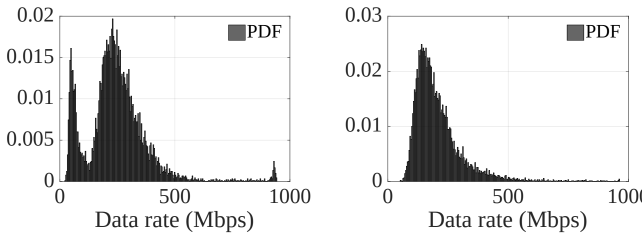
[5]Note that it is possible that the network traffic may not follow a log-normal distribution at very fine or coarse aggregation granularities.

[6]Due to lack of space, Q-Q plots are not included as we would have to present plots for each trace, separately.



(a) CAIDA traces (b) Waikato traces (c) Auckland traces (d) Twente traces (e) MAWI traces

Fig. 2: Normalised Log-Likelihood Ratio (LLR) test results for all studied traces and log-normal distribution. Aggregation timescales are 5 sec, 1 sec, 100 msec and 5 msec. Circled points in the plot are the ones with p-value greater than 0.1, i.e. likelihood test is inconclusive with respect to fitting the log-normal distribution to the traffic data.



(a) Anomalous trace (b) Log-normal trace

Fig. 3: PDF of an anomalous and non-anomalous trace.

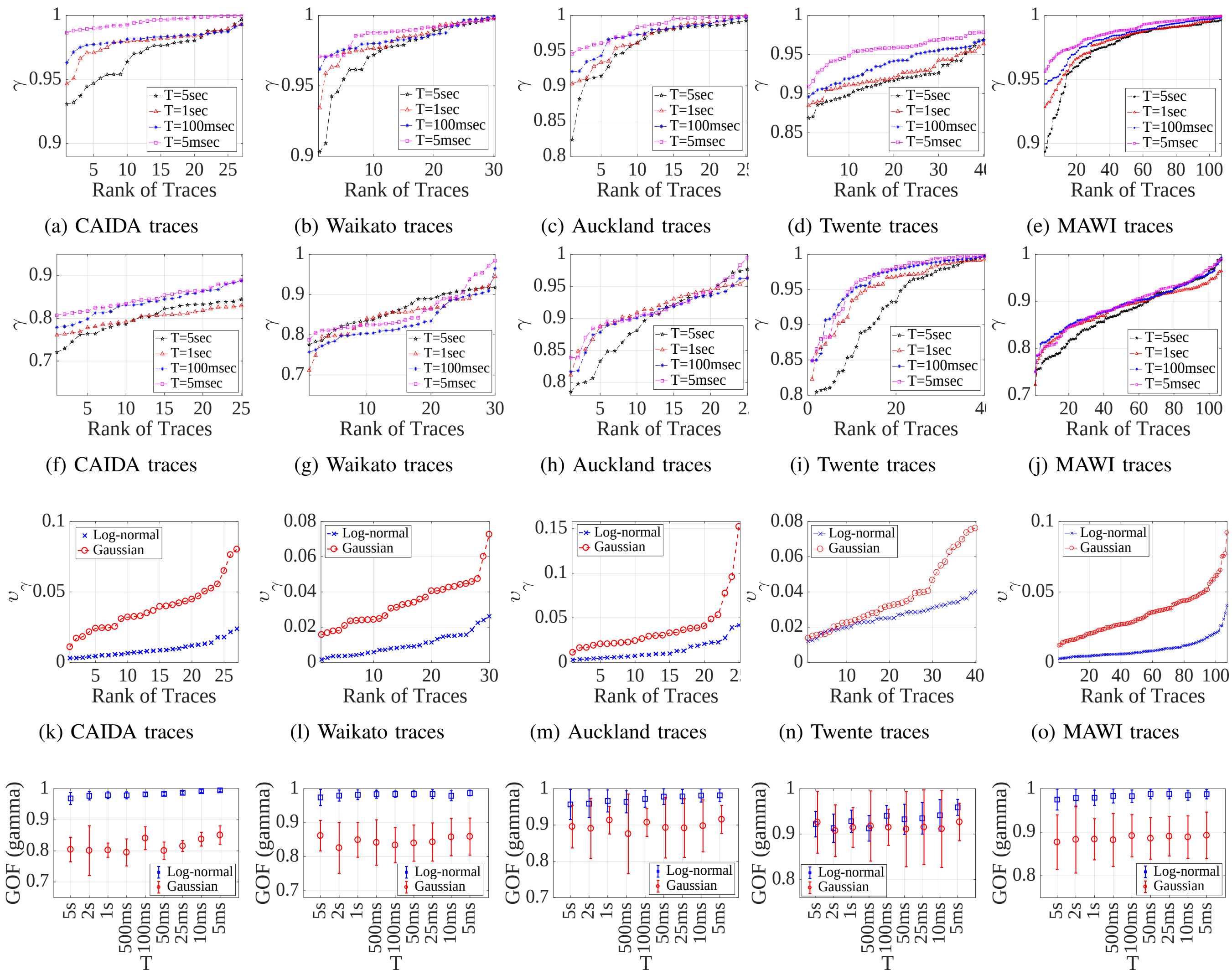
C. *Fitting the log-normal and Gaussian distributions using the correlation coefficient test*

The linear correlation coefficient test has been widely used to assess the fit of a distribution to empirical data. To reinforce the results of Section III-A, we employ the linear correlation coefficient assuming that the log-normal distribution is the best fit (as we showed in Section III). We compare the results of this test for both the log-normal and Gaussian distributions. We use the linear correlation coefficient as defined in [19]:

$$\gamma = \frac{\sum_{i=1}^{n}\left(S_{(i)}-\hat{\mu}\right)\left(x_i-\hat{x}\right)}{\sqrt{\sum_{i=1}^{n}\left(S_{(i)}-\hat{\mu}\right)^2 . \sum_{i=1}^{n}\left(x_i-\hat{x}\right)^2}} \quad (1)$$

where $S_{(i)}$ is the observed sample i, and $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} S_{(i)}$ is the samples' mean value. x_i is sample i from the reference distribution (log-normal in our case), which can be calculated from the inverse CDF of the reference random variable $x_i = F^{-1}\left(\frac{i}{n+1}\right)$ and $\hat{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the respective mean value. The value of the correlation coefficient can vary between $-1 \leq \gamma \leq 1$, with a 1, 0 and -1 indicating perfect correlation, no correlation and perfect anti-correlation, respectively. Strong goodness-of-fit (GOF) is assumed to exist when the value of γ is greater than 0.95 [17].

We measure the linear correlation coefficient for all datasets at four different aggregation timescales (ranging from 5 msec to 5 sec) and plot the results in Figures 4(a) to 4(e) for the log-normal distribution and Figures 4(f) to 4(j) for the Gaussian distribution. Traces are ordered by the value of γ for the given timescale. It can be clearly seen that $\gamma > 0.95$ for most traces when employing the test for the log-normal distribution, but this is not the case for the Gaussian distribution. γ is larger for smaller aggregation timescales indicating that the log-normal distribution is an even better fit as the aggregation gets finer. For very small values of T, i.e. lower than 1 msec, data samples exhibit binary behaviour, where either a packet is transmitted or not during each examined time frame [18]. We have examined γ for very short (and large) aggregation timescales, and can confirm the absence of a model describing the data (for brevity, we have omitted the relevant figures).



(a) CAIDA traces (b) Waikato traces (c) Auckland traces (d) Twente traces (e) MAWI traces

(f) CAIDA traces (g) Waikato traces (h) Auckland traces (i) Twente traces (j) MAWI traces

(k) CAIDA traces (l) Waikato traces (m) Auckland traces (n) Twente traces (o) MAWI traces

(p) CAIDA traces (q) Waikato traces (r) Auckland traces (s) Twente traces (t) MAWI traces

Fig. 4: Correlation coefficient test results for all studied traces and different timescales.

Next, we calculate υ_γ (the variation of γ) for each dataset. υ_γ gives an indication of the stability of γ for each dataset, for all timescales tested. This metric is defined as:

$$\upsilon_\gamma = \sqrt{var(\gamma_{T_1}, \gamma_{T_2}, \gamma_{T_3}, \gamma_{T_4})} \tag{2}$$

where $T_1 = 5$ sec, $T_2 = 1$ sec, $T_3 = 100$ msec and $T_4 = 5$ msec. Figures 4(k) to 4(o) show the results for each dataset with the traces ranked by υ_γ. For log-normal model, υ_γ is very small (below 0.045) for all traces, therefore we can conclude that γ is almost constant for all studied aggregation timescales. While υ_γ is higher for Gaussian model. Furthermore, the error bars in Figures 4(p) to 4(t) represent the standard deviation of the correlation coefficient at different timescales (see x-axis). This again shows that for log-normal model γ is larger than 0.95 (at different T values) for most CAIDA and MAWI traces, while it is larger than 0.9 for all other datasets. This is not the case with the Gaussian model, where most γ values are less than 0.9.

Overall, the correlation coefficient test reinforces the results extracted in Section III-A, providing strong evidence that the log-normal distribution is the best fit for all studied traces. Superior performance of our model can also be seen from comparison of our results for correlation coefficient with those in [20] where the Gaussian model was used.

IV. Bandwidth Provisioning

It has been previously suggested that network link provisioning could be based on fitted traffic models instead of relying on straightforward empirical rules [20]. In this way, over- or under-provisioning can be mitigated or eliminated even in the presence of strong traffic fluctuations. Such approaches rely on having a statistical model that accurately describes the network traffic. This is therefore an excellent area for applying our findings on fitting the log-normal distribution to Internet traffic data. In the literature, the following inequality (the authors call it the “link transparency formula”) has been used for bandwidth provisioning [18]:

$$P\left(A(T) \geq CT\right) \leq \varepsilon. \tag{3}$$

In words, this inequality states that the probability that the captured traffic $A(T)$ over a specific aggregation timescale T is larger than the link capacity has to be smaller than the value of a performance criterion ε. The value of ε is chosen carefully by the network provider in order to meet a specific SLA [20]. Likewise, the value of the aggregation time T should be sufficiently small so that the fluctuations in the traffic can be modelled as well, taking into account the buffering capabilities of network switching devices[7].

We compare bandwidth provisioning using Meent's approximation formula [20] (assuming Gaussian) and using a log-normal traffic model.

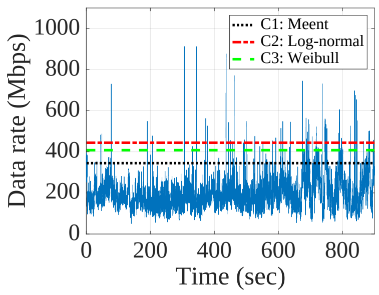


Fig. 5: Data rate of a MAWI trace ($T = 100$ msec and $\varepsilon = 0.01$). The horizontal lines represent the calculated link capacity based on different models.

A. *Bandwidth provisioning using Meent's formula*

To find the minimum required link capacity, Meent et al. [20] proposed a bandwidth provisioning approach that is based on the assumption that the traffic follows a Gaussian distribution. Meent's dimensioning formula is defined as follows [20]:

$$C1 = \mu + \frac{1}{T}\sqrt{-2log(\varepsilon).v(T)} \tag{4}$$

where μ is the average value of the traffic, $v(T)$ is the variance at timescale T and ε is the performance criterion. The link capacity is obtained by adding a safety margin value

$$\text{Safety margin} = \sqrt{-2log(\varepsilon)}\ .\ \sqrt{\frac{v(T)}{T^2}}$$

to the average of the captured traffic (see Equation 4). This safety margin value depends on ε and the ratio $\sqrt{v(T)/T^2}$. As the value of ε decreases the safety margin increases. For example, when the value of ε decreases from 10^{-2} to 10^{-4}, then value of the safety margin increases by 40%. This is different from conventional link dimensioning methods, where the safety margin is fixed to be 30% above the average of the presented traffic [20], [21]. Traffic tails are represented using the Chernoff bound, as follows:

$$P\left(A(T) \geq CT\right) \leq e^{-SCT} E\left[e^{SA(T)}\right]. \tag{5}$$

Here $E\left[e^{SA(T)}\right]$ is the moment generation function (MGF) of the captured traffic $A(T)$.

B. *Bandwidth provisioning based on the log-normal model*

Here we investigate whether we could achieve more reliable bandwidth provisioning by adopting the log-normal traffic model. We calculate the mean and variance from the captured trace and generate the respective log-normal model. Then, we use the CDF function (F) to solve the link transparency formula shown in Equation 3. Hence, F is defined as $F(C) = P(A(T)/T < C)$, which can be solved to find C, as follows:

$$C2 = F^{-1}\left(1 - \varepsilon\right). \tag{6}$$

C. *Comparison of bandwidth provisioning approaches*

In this section, we compare the bandwidth provisioning approaches described above. The performance indicator is the empirical value of the performance criterion, which is denoted by $\hat{\varepsilon}$ and defined as follows:

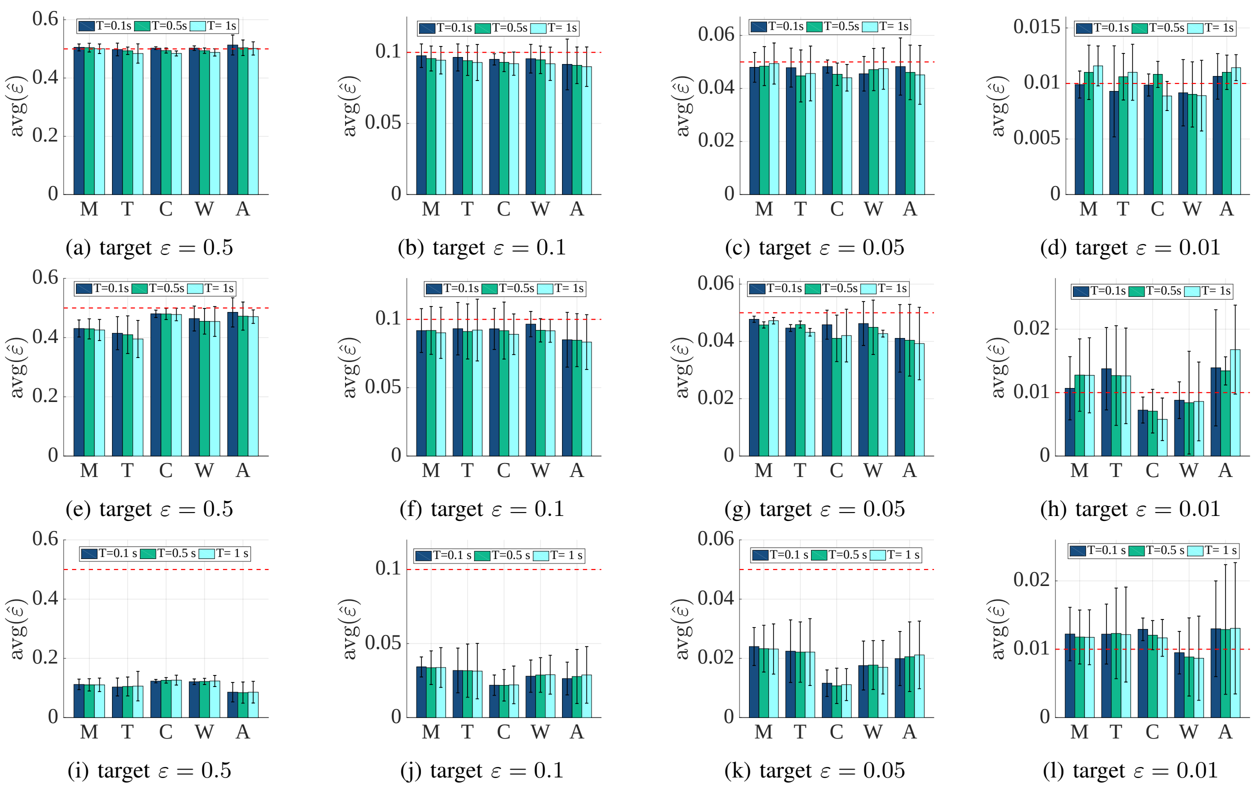
$$\hat{\varepsilon} = \frac{\#\left\{A_i | A_i \geq CT\right\}}{n}\ , i \in 1 \ldots n. \tag{7}$$

In words, this empirical value is the percentage of all the data samples of the captured traffic which are measured larger than the estimated link capacity. Ideally, $\hat{\varepsilon}$ would be equal to the target value of the performance criterion ε. The difference between $\hat{\varepsilon}$ and ε is due to the fact that the chosen traffic model is not accurately describing the real network traffic. A simple example of the described comparison approach is illustrated in Figure 5, in which we plot the captured data rate for a MAWI trace ($T = 100$ msec)[8]. The calculated capacity values from each approach when the target ε is 0.01 are $C1 = 344.8$ Mbps and $C2 = 444.3$ Mbps (represented by the horizontal lines in Figure 5). The empirical value can be calculated by using Equation 7, which gives $\hat{\varepsilon}_1 = 0.042$ and $\hat{\varepsilon}_2 = 0.012$. Obviously, with the first approach the network operator would not be able to meet the target $\varepsilon = 0.01$, while with the second approach the empirical value is close to the target.

We next compare results of bandwidth provisioning calculations based on the (a) Meent's formula, (b) Weibull model and (c) proposed log-normal model. Figure 6(a)-(d) shows the average of the empirical value (avg($\hat{\varepsilon}$)) for all traces in each dataset at $T = 0.1$ sec, $T = 0.5$ sec and $T = 1$ sec. The value of T is chosen to be sufficiently small so that the fluctuations in the traffic can be modelled as well. Each model is tested for four different values of the performance

[7]Large traffic fluctuations at very short aggregation timescales are smoothed by the presence of buffers at network routers and switches.

[8]Note that in all subsequent figures we have also included results for a Weibull model to get insights about bandwidth provisioning using a heavy-tailed distribution.



(a) target $\varepsilon = 0.5$ (b) target $\varepsilon = 0.1$ (c) target $\varepsilon = 0.05$ (d) target $\varepsilon = 0.01$

(e) target $\varepsilon = 0.5$ (f) target $\varepsilon = 0.1$ (g) target $\varepsilon = 0.05$ (h) target $\varepsilon = 0.01$

(i) target $\varepsilon = 0.5$ (j) target $\varepsilon = 0.1$ (k) target $\varepsilon = 0.05$ (l) target $\varepsilon = 0.01$

Fig. 6: Link dimensioning based on (a-d) log-normal model, (e-h) Weibull model and (i-l) Meent's formula: avg($\hat{\varepsilon}$) for different datasets (M: MAWI, T: Twente, C: CAIDA, W: Waikato, A: Auckland), aggregation timescales (100 msec, 500 msec and 1 s), and target values of ε (0.5, 0.1, 0.05 and 0.01). Error bars represent stderr $|\varepsilon - \hat{\varepsilon}|$.

criterion: $\varepsilon = 0.5$, $\varepsilon = 0.1$, $\varepsilon = 0.05$ and $\varepsilon = 0.01$. In Figure 6(a)-(d) we clearly see that the log-normal model is able to satisfy the required performance criterion ε at different aggregation time-scales for all datasets. In contrast, Meent's formula failed to allocate sufficient bandwidth, which results in missing the target performance criterion ε for all datasets and target performance values, as depicted in Figure 6(i)-(l) (see horizontal red line). The Weibull distribution performs better comparing to Meent's formula, but bandwidth provisioning using the log-normal model is far superior, as can be seen from Figures 6(a)-(d) and 6(e)-(h).

V. 95th Percentile Pricing Scheme Based on Log-Normal Model

Traffic billing is typically based on the 95th percentile method [22]. Traffic volume is measured at border network devices (typically aggregated at time intervals of 5 minutes) and bills are calculated according to the 95-percentile of the distribution of measured volumes; i.e. network operators calculate bills by disregarding occasional traffic spikes. Forecasting future bills, which is important for ISPs and clients, can be done using a model of the traffic calculated through previously sampled traffic. In this section, we apply our findings on Internet traffic modelling in predicting the cost of traffic according to the 95th percentile method. For each network trace we calculate the actual 95th percentile of the traffic volume. The majority of the studied traffic traces were 15-minute long but operators typically use measurements traffic volumes for much longer periods, therefore we scale down the calculation of the 95th percentile by dividing each trace (900 seconds) into 90 groups (10 seconds length each). The authors appreciate that by using 15-minute rather than day long traces we omit any study of diurnal effects in the distribution. We note that the sum of several log-normal distributions is itself very accurately represented by a log-normal distribution [23]. Hypothetically, therefore, if 96 consecutive 15-minute traces fit a log-normal distribution (with different parameters for each) then the resulting 24 hour trace is also likely to be a good fit to a log-normal. We also note that the distributions tested were on a level playing field in that they would all be affected equally by the shorter duration of the data sets.

We calculate the 95th percentile for the observed traffic. We then fit a Gaussian, Weibull and log-normal distribution to each trace (for $T = 100$ msec) and calculate the 95th percentile of the fitted distribution. We plot the actual 95th percentile against the three predictions in Figure 7 with a red reference line to show where perfect predictions would be located. It is clear that the log-normal model provides much more accurate predictions of the 95th percentile than the Gaussian model. As with the bandwidth dimensioning case discussed in Section IV,

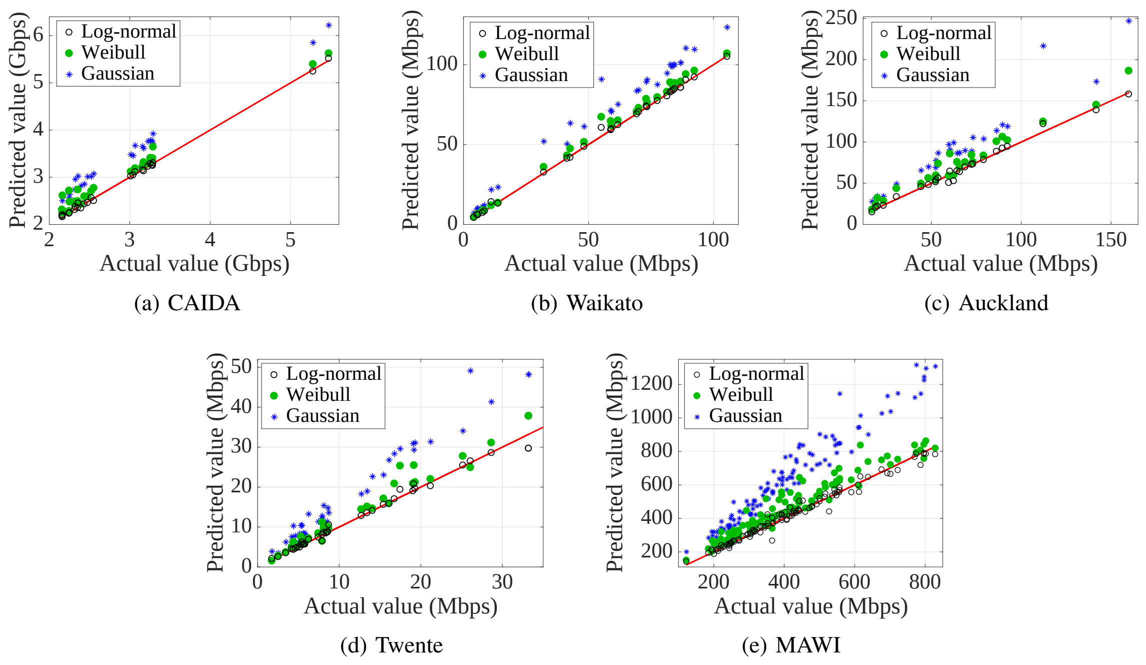


Fig. 7: 95th percentile values (actual vs predicted rates) based on log-normal, Weibull and Gaussian models. An ideal model would result in points in the plot area that fall exactly on the red line.

TABLE I: Goodness of fit (GOF) using normalised root mean squared error (NRMSE)

Model/Dataset	CAIDA	Waikato	Auckland	Twente	MAWI
Log-normal	0.0399	0.0401	0.1058	0.0979	0.1528
Weibull	0.2410	0.1148	0.2984	0.2123	0.4145
Gaussian	0.5544	0.4193	0.6866	0.5741	0.9828

the Weibull is better than the Gaussian model but worse than the proposed log-normal model.

We employ the normalised root mean squared error (NRMSE) as a goodness of fit to the results in Figure 7. NRMSE measures the differences between values predicted by a hypothetical model and the actual values. In other words, it measures the quality of the fit between the actual data and the predicted model. Table I shows the NRMSE for all datasets and the three considered models. It is clear that the lowest NRMSE value is for the log-normal model, which is the best model compared to the Gaussian and Weibull ones.

VI. Related work

Reliable traffic modelling is important for network planning, deployment and management; e.g. for traffic billing and network dimensioning. Historically, network traffic has been widely assumed to follow a Gaussian distribution. In [5], [7], the authors studied network traces and verified that the Gaussianity assumption was valid (according to simple goodness-of-fit tests they used) at two different timescales. In [24], the authors studied traffic traces during busy hours over a relatively long period of time and also found that the Gaussian distribution is a good fit for the captured traffic. Schmidt et al. [8] found that the degree of Gaussianity is affected by short and intensive activities of single network hosts that create sudden traffic bursts. All the above mentioned works agreed on the Gaussian or 'fairly Gaussian' traffic at different levels of aggregations in terms of timescale and number of users. The authors in [19], [25] examined the levels of aggregation required to observe Gaussianity in the modelled traffic, and concluded that this can be disturbed by traffic bursts. The work in [9], [26] reinforces the argument above, by showing existence of large traffic spikes at short timescales which result in high values in the tail. Compared to existing literature, our findings are based on a modern, principled statistical methodology, and traffic traces that are spatially and temporally diverse. We have tested several hypothesised distributions and not just Gaussianity.

An early work drawing attention to the presence of heavy tails in Internet file sizes (not traffic) is that of Crovella and Bestavros [2]. Deciding whether Internet flows could be heavy-tailed became important as this implies significant departures from Gaussianity. The authors in [27] provided robust evidence for the presence of various kinds of scaling, and in particular, heavy-tailed sources and long range dependence in a large dataset of traffic spanning a duration of 14 years.

Understanding the traffic characteristics and how these evolve is crucial for ISPs for network planning and link dimensioning. Operators typically over-provision their networks. A common approach to do so is to calculate the average bandwidth utilisation [6] and add a safety margin. As a rule of thumb, this margin is defined as a percentage of the calculated bandwidth utilisation [21]. Meent et al. [20] proposed a new bandwidth provisioning formula, which calculates the minimum bandwidth that guarantees the required performance, according to an underlying SLA. This approach relies on the statistical parameters of the captured traffic and a performance parameter. The underlying fundamental assumption for this to work is that the traffic the network operator sees follows a Gaussian distribution. Same approach has been used in [18].

The 95th percentile method is used widely for network traffic billing. Dimitropoulos et al. [22] have found that the computed 95th percentile is significantly affected by traffic aggregation parameters. However, in their approach they do not assume any underlying model of the traffic; instead, they base their study on specific captured traces. Stanojevic et al. [4] proposed the use of Shapley value for computing the contribution of each flow to the 95th percentile price of interconnect links. Works [28]–[31] propose calculating the 95th percentile using experimental approaches. Xu et al. [32] assume that network traffic follows a Gaussian distribution"through reasonable aggregation" and propose a cost efficient data centre selection approach based on the 95th percentile.

VII. Conclusion

The distribution of traffic on Internet links is an important problem that has received relatively little attention. We use a well-known, state-of-the-art statistical framework to investigate the problem using a large corpus of traces. The traces cover several network settings including home user access links, tier 1 backbone links and campus to Internet links. The traces are from times from 2002 to 2018 and are from a number of different countries. We investigated the distribution of the amount of traffic observed on a link in a given (small) aggregation period which we varied from 5 msec to 5 sec. The hypotheses compared were that the traffic volume was heavy-tailed, that the traffic was log-normal and that the traffic was normal (Gaussian). The vast majority of traces fitted the log-normal assumption best and this remained true all timescales tried. Where no distribution tested was a good fit this could be attributed either to the link being saturated (at capacity) for a large part of the observation or exhibiting signs of link-failure (no or very low traffic for part of the observation).

We investigate the impact of the distribution on two sample traffic engineering problems. Firstly, we looked at predicting the proportion of time a link will exceed a given capacity. This could be useful for provisioning links or for predicting when SLA violation is likely to occur. Secondly, we looked at predicting the 95th percentile transit bill that ISP might be given. For both of these problems the log-normal distribution gave a more accurate result than a heavy-tailed distribution or a Gaussian distribution. We conclude that the log-normal distribution is a good (best) fit for traffic volume on a normally functioning internet links in a variety of settings and over a variety of timescales, and further argue that this assumption can make a large difference to statistically predicted outcomes for applied network engineering problems.

In future work, we plan to test the stationarity of the traffic traces.

References

[1] P. Pruthi and A. Erramilli, "Heavy-tailed on/off source behavior and self-similar traffic," in *Proc. of ICC*, 1995.

[2] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM ToN*, 1997.

[3] P. Loiseau, P. Goncalves, G. Dewaele, P. Borgnat, P. Abry, and P. V. B. Primet, "Investigating self-similarity and heavy-tailed distributions on a large-scale experimental facility," *IEEE/ACM ToN*, 2010.

[4] R. Stanojevic and et. al., "On economic heavy hitters: Shapley value analysis of 95th-percentile pricing," in *Proc. of ACM IMC*, 2010.

[5] R. V. D. Meent, M. Mandjes, and A. Pras, "Gaussian traffic everywhere?" in *Proc. of IEEE ICC*, 2006.

[6] R. d. O. Schmidt, H. van den Berg, and A. Pras, "Measurement-based network link dimensioning," in *Proc. of IFIP/IEEE*, 2015.

[7] R. d. O. Schmidt, R. Sadre, and A. Pras, "Gaussian traffic revisited," in *Proc. of IFIP Networking*, 2013.

[8] R. d. O. Schmidt, R. Sadre, N. Melnikov, J. Schnwlder, and A. Pras, "Linking network usage patterns to traffic gaussianity fit," in *Proc. of IFIP Networking*, 2014.

[9] X. Yang, "Designing traffic profiles for bursty Internet traffic," in *Proc. of IEEE GLOBECOM*, 2002.

[10] A. Clauset, C. S. Rohilla, and M. Newman, "Power-law distributions in empirical data," *arXiv:0706.1062v2*, 2009.

[11] "The caida ucsd anonymized internet traces," 2016. [Online]. Available: http://www.caida.org/data/passive/passive_dataset.xml

[12] "Mawi archive," 2018. [Online]. Available: http://mawi.wide.ad.jp/

[13] R. R. R. Barbosa, R. Sadre, A. Pras, and R. van de Meent, "Simpleweb/university of twente traffic traces data repository," http://eprints.eemcs.utwente.nl/17829/, Tech. Rep., 2010.

[14] "Wits: Waikato internet traffic storage," 2013. [Online]. Available: https://wand.net.nz/wits/waikato/8/

[15] "Wits: Auckland x," 2009. [Online]. Available: https://wand.net.nz/wits/auck/10/

[16] J. Alstott, E. Bullmore, and D. Plenz, "powerlaw: a python package for analysis of heavy-tailed distributions," *arXiv:1305.0215*, 2014.

[17] M. Mandjes and R. van de Meent, "Resource dimensioning through buffer sampling," *IEEE/ACM Transactions on Networking*, 2009.

[18] R. d. O. Schmidt, R. Sadre, A. Sperotto, H. van den Berg, and A. Pras, "Impact of packet sampling on link dimensioning," *IEEE Transactions on Network and Service Management*, 2015.

[19] J. Kilpi and I. Norros, "Testing the gaussian approximation of aggregate traffic," in *Proc. of SIGCOMM*, 2002.

[20] A. Pras, L. Nieuwenhuis, R. van de Meent, and M. Mandjes, "Dimensioning network links: a new look at equivalent bandwidth," *IEEE Network*, 2009.

[21] "Best practices in core network capacity planning," online, accessed July 2018. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/wan-automation-engine/white_paper_c11-728551.pdf

[22] X. Dimitropoulos, P. Hurley, A. Kind, and M. P. Stoecklin, "On the 95-Percentile Billing Method," in *Proc. of PAM*, 2009.

[23] R. Mitchell, "Permanence of the log-normal distribution." *J. Optical Society of America*, 1968.

[24] J. L. García-Dorado, J. A. Hernández, J. Aracil, J. E. López de Vergara, and S. Lopez-Buedo, "Characterization of the busy-hour traffic of IP networks based on their intrinsic features," *Computer Networks*, 2011.

[25] A. B. Downey, "Evidence for Long-tailed Distributions in the Internet," in *Proc. of ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[26] H. Abrahamsson, B. Ahlgren, P. Lindvall, J. Nieminen, and P. Tholin, "Traffic characteristics on 1gbit/s access aggregation links," in *Proc. of IEEE ICC*, 2017.

[27] R. Fontugne and et. al., "Scaling in internet traffic: A 14 year and 3 day longitudinal study, with multiscale analyses and random projections," *IEEE/ACM Transactions on Networking*, 2017.

[28] L. Golubchik and et. al., "To send or not to send: Reducing the cost of data transmission," in *Proc. of IEEE INFOCOM*, 2013.

[29] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *Proc. of ACM SIGCOMM*, 2011.

[30] I. Castro, R. Stanojevic, and S. Gorinsky, "Using Tuangou to Reduce IP Transit Costs," *IEEE/ACM Transactions on Networking*, 2014.

[31] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. of IEEE INFOCOM*, 2013.

[32] ——, "Cost efficient datacenter selection for cloud services," in *Proc. of IEEE ICCC*, 2012.

Chapter 6

Conclusion and Future Directions

This chapter presents the contributions and conclusions for the research presented in this thesis and suggests some directions for further research.

6.1 Contributions and Conclusions

The papers presented in Chapters 3, 4 and 5 form the main contributions in this thesis. The first research question in this thesis is about designing a data transport protocol that supports diverse and modern communication patterns in data centres while providing high goodput, low completion times and high network utilisation to a diverse set of data centre applications. This research question is answered in the first two papers, where we present the design of SCDP, a general-purpose data transport protocol for data centres [9, 10]. SCDP is the first protocol that provides native support for one-to-many and many-to-one communication in data centres. It also ensures good performance for long and short unicast flows. SCDP integrates RaptorQ codes, receiver-driven data transport, in-network packet trimming and MLFQ. RaptorQ codes incur some minimal network overhead, only when loss occurs in the network, but our experimental evaluation showed that this is negligible compared to the significant performance benefits of supporting one-to-many and many-to-one workloads. RaptorQ codes also incur computational overhead and associated latency when a loss occurs. However, we show that this is rare for short flows because of MLFQ. For long flows, block pipelining alleviates the problem by splitting large blocks into smaller ones and decoding each of these smaller blocks while retrieving the next one. As a result, latency is incurred only for the last smaller block. RaptorQ codes have been shown to perform at line speeds even on a single core; we expect that with hardware offloading the overall overhead will not be significant. Our extensive evaluation of SCDP shows substantial performance improvements over other data transport protocols. In the third paper [7], we present our developed simulation framework for data centre protocols in OMNeT++. This framework is used in the evaluations in the first two papers.

The second research question in this thesis is about understanding and modelling the Internet traffic volumes by investigating real Internet traces. The modelling process has to follow a well-established statistical methodology. This research question is answered in the fourth paper [8], where we present a very comprehensive statistical analysis of aggregated Internet traffic volumes over a large set of public datasets. These datasets span a diverse range of locations, time scales, link speeds and user populations. The goal of the work is to derive the marginal distribution of the time discrete process X (volume of traffic in bit/s) where x_i is the volume measured on a pre-defined time scale going from 1 ms to 5 sec. The distribution of traffic on Internet links is an important problem that has received relatively little attention. We use a well-known, state-of-the-art statistical framework to investigate the problem using a large corpus of traces. The conclusion is that the best fit for the marginal distribution is a log-normal distribution, compared to Gaussian, Weibull, exponential or power law. We investigate the impact of the distribution on two sample traffic engineering problems. Firstly, we looked at predicting the proportion of time a link will exceed a given capacity. This could be useful for provisioning links or for predicting when SLA violation is likely to occur. Secondly, we looked at predicting the 95th percentile transit bill that ISP might be given. For both of these problems, the log-normal distribution gave a more accurate result than a heavy-tailed distribution or a Gaussian distribution.

6.2 Future research

The work in this thesis fits in the broader context of the new challenges and opportunities that have arisen due to the significant interest and investment in large-scale data centres. Data centres provide an opportunity to revisit some of the fundamental issues of packet switched networks, such as congestion control, forwarding/routing and traffic engineering. This makes data centres a very rich environment for networking researchers. We conclude the thesis by discussing some of the directions in which our work can be pursued further.

- Our proposed transport protocol SCDP [10] is agnostic to the underlying data centre network topology i.e., it is deployable in both switch-centric and server-centric data centre topologies. In our evaluation, we adopted a switch-centric Fat-tree topology. Thus, as a future work we plan to evaluate SCDP in server-centric topologies as DCell [87] and BCube [85]. We expect SCDP to outperform other modern transport protocols in this type of topologies as ordering does not harm the performance of SCDP.

- SCDP flow and congestion control mechanism is based on packet trimming and pacing pull requests. It would be interesting to investigate other mechanisms that can benefit from RaptorQ proprieties to design a reliable data transport protocol. Furthermore, there are some switch vendors who are interesting in implementing NDP switch[1], and as SCDP adopts NDP switch, we would be looking forward to

[1]Private communication with Professor Mark Handley, also mentioned here [92]

seeing this happens. Moreover, we have not evaluated the coexistence between SCDP and TCP flows, however, this can be ensured by scheduling TCP flows to a separate queue and performing fair queueing between the two types of flows. Also, we plan to build more OMNeT++/INET-based models that simulate other modern data transport protocols for data centre networks, such as MPTCP [157] and PIAS [24].

- We built a RaptorQ model in Matlab and C++ based on the specifications in RFC6330 [67], however, there is a space of optimisation in these implementations or one can get access to an optimised RaptorQ model by Codornices [2].

- Currently, SCDP is implemented in OMNeT++, a packet-level discrete-event simulator. We plan to have a prototype of SCDP in real systems, for example, Linux end-systems using DPDK [171], hardware switch using NetFPGA [144] and P4 switch [30].

- In our fourth paper [8], the distribution tests assume that the underlying data is strictly stationary. The paper does not test for stationarity or discuss deviations from stationarity in the measurement dataset. We plan to test a timescale within which the stationarity assumption and the log-normal distribution fits hold. We have progressed in this by using some well-known stationarity tests such as ADF [142] and KPSS [111]. The results of running these tests on the datasets will be published soon. Also, this work may be extended to another direction to do analysing and modelling of packet inter-arrival times in the datasets. Furthermore, the used datasets in this work can be extended to include any available data centres traces. Finally, testing the time correlation property would be interesting as well.

[2]https://www.codornices.info/

References

[1] C. Clos. A. A Study of Non-blocking Switching Networks. *Bell System Technical Journal*, 1953. 10

[2] Martin Abadi and et. al. TensorFlow: A system for large-scale machine learning. In *Proc. of USENIX (OSDI)*, 2016. 19

[3] H. Abrahamsson, B. Ahlgren, P. Lindvall, J. Nieminen, and P. Tholin. Traffic characteristics on 1Gbit/s access aggregation links. In *Proc. of IEEE ICC*, 2017. 4, 41, 42

[4] Akka. Akka Build Powerful Reactive Concurrent and Distributed Applications more Easily Using UDP. `https://doc.akka.io/docs/akka/2.5.4/java/io-udp.html`, 2019. [Online; accessed 19-July-2019]. 2

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proc. of SIGCOMM*, 2008. ix, 9, 10, 40

[6] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. of USENIX*, 2010. 3, 13, 19, 24

[7] Mohammed Alasmar and George Parisis. Evaluating modern data centre transport protocols in OMNeT++/INET. In *Proc. of the 6th OMNeT++ Community Summit Conference*, Hamburg, Germany, 2019. v, vi, 6, 7, 8, 67, 90

[8] Mohammed Alasmar, George Parisis, Richard Clegg, and Nickolay Zakhleniuk. On the Distribution of Traffic Volumes in the Internet and its Implications. In *Proc. of INFOCOM*, 2019. v, vi, 6, 7, 8, 43, 80, 91, 92

[9] Mohammed Alasmar, George Parisis, and Jon Crowcroft. Polyraptor: Embracing Path and Data Redundancy in Data Centres for Efficient Data Transport. In *Proc. of the ACM SIGCOMM Conference on Posters and Demos*, 2018. v, 7, 40, 48, 90

[10] Mohammed Alasmar, George Parisis, and Jon Crowcroft. SCDP: Systematic Rateless Coding for Efficient Data Transport in Data Centres. Submitted to IEEE/ACM

Transactions on Networking, 2019. https://arxiv.org/abs/1909.08928. v, 5, 7, 11, 13, 30, 48, 90, 91

[11] Mohammed Alasmar and Nickolay Zakhleniuk. Network Link Dimensioning based on Statistical Analysis and Modeling of Real Internet Traffic. Submitted to IEEE/ACM Transactions on Networking, 2017. https://arxiv.org/abs/1710.00420. 41

[12] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman. Data center transport mechanisms: Congestion control theory and IEEE standardization. In *Annual Allerton Conference on Communication, Control, and Computing*, 2008. 40

[13] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *Proc. of SIGCOMM*, 2010. 3, 5, 14, 17, 19, 20, 22, 26, 28, 29

[14] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *Proc. of NSDI USENIX*, 2012. 12

[15] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proc. of SIGCOMM*, 2013. 3, 12, 19, 22, 23, 29

[16] M Allman, V Paxson, and E Blanton. TCP Congestion Control. *IETF, RFC 5681*, 2009. 2, 14

[17] Alexey Andreyev. Introducing data center fabric, the next-generation Facebook data center network. Online; accessed August, 2019. https://tinyurl.com/tt685av. 1, 11

[18] Apache. Apache Giraph. Online; accessed August, 2019. https://giraph.apache.org/. 19

[19] Apache. Apache HTTP Server Project. Online; accessed August, 2019. https://www.mysql.com/. 19

[20] Apache. Apache Storm. Online; accessed August, 2019. https://storm.apache.org/. 19

[21] Apache. HDFS. Online; accessed August, 2019. https://hadoop.apache.org/. 2, 18

[22] M. Bagnulo. Threat analysis for TCP extensions for multipath operation with multiple addresses. *IETF, RFC 6181*, 2011. 20

[23] M. Bagnulo, C. Paasch, F. Gont, O. Bonaventure, and C. Raiciu. Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP). *IETF, RFC 7430*, 2015. 21

[24] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *Proc. of NSDI, USENIX*, 2015. 5, 14, 19, 23, 29, 92

[25] L. Baldantoni, H. Lundqvist, and G. Karlsson. Adaptive end-to-end FEC for improving TCP performance over wireless links. In *Proc. of IEEE International Conference on Communications*, 2004. 27

[26] Theophilus Benson, Aditya Akella, and David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. of IMC*, 2010. 30

[27] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding Data Center Traffic Characteristics. In *Proc. of SIGCOMM*, 2010. 30, 31

[28] Debopam Bhattacherjee, Waqar Aqeel, Ilker Nadi Bozkurt, Anthony Aguirre, Balakrishnan Chandrasekaran, P. Brighten Godfrey, Gregory Laughlin, Bruce Maggs, and Ankit Singla. Gearing Up for the 21st Century Space Race. In *Proc. of HotNets*, 2018. 1

[29] Sanjit Biswas, John Bicket, Edmund Wong, Raluca Musaloiu-E, Apurv Bhartia, and Dan Aguayo. Large-scale Measurements of Wireless Network Behavior. In *Proc. of SIGCOMM*, 2015. 1

[30] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-independent Packet Processors. In *Proc. of SIGCOMM*, 2014. 5, 11, 12, 14, 26, 92

[31] Timm Böttger, Gianni Antichi, Eder Leão Fernandes, Roberto di Lallo, Marc Bruyere, Steve Uhlig, and Ignacio Castro. The Elusive Internet Flattening: 10 Years of IXP Growth. *ArXiv*, abs/1810.10963, 2018. 1

[32] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. Open Connect Everywhere: A Glimpse at the Internet Ecosystem Through the Lens of the Netflix CDN. In *Proc. of SIGCOMM*, 2018. 1

[33] Robert T. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, 1989. 29

[34] L Brakmo and L Peterson. TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 1995. 14, 26, 28

[35] I. Castro, R. Stanojevic, and S. Gorinsky. Using Tuangou to Reduce IP Transit Costs. *IEEE/ACM Transactions on Networking*, 22(5), 2014. 43

[36] Li Chen, Shuihai Hu, Kai Chen, Haitao Wu, and Danny H. K. Tsang. Towards minimal-delay deadline-driven data center TCP. In *Proc. of HotNets*, 2013. 29

[37] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization. In *Proc. of SIGCOMM*, 2018. 3

[38] Yanpei Chen, Rean Griffith, Junda Liu, and Anthony Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Proc. of SIGCOMM*, 2009. 3, 5, 14, 15, 29

[39] Yanpei Chen, Rean Griffith, David Zats, Anthony D. Joseph, and Randy Katz. Understanding TCP incast and its implications for big data workloads. In *Proc. of USENIX*, 2012. 3, 5, 15

[40] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. In *Proc. of USENIX*, 2014. 5

[41] Mosharaf Chowdhury and Ion Stoica. Coflow: A Networking Abstraction for Cluster Applications. In *Proc. of HotNets*, 2012. ix, 19

[42] CISCO. Best Practices in Core Network Capacity Planning. Online; accessed August, 2019. https://tinyurl.com/wk4ge3t. 4

[43] C.K.P. Clarke. R&D White Paper: Reed-Solomon error correction. *Research & Development British Broadcasting Corporation*, 2002. 32

[44] Aaron Clauset, Cosma Shalizi Rohilla, and M Newman. Power-law distributions in empirical data. *arXiv:0706.1062v2*, 2009. vi, 4, 6, 43, 44, 80

[45] James Corbett and et. al. Spanner: Google's Globally-distributed Database. In *Proc. of OSDI/USENIX*, 2012. 18

[46] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997. 4

[47] Wenzhi Cui and Chen Qian. Dual-structure Data Center Multicast using Software Defined Networking. arxiv, 2014. http://arxiv.org/abs/1403.8065. 2

[48] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang. FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol. *IEEE/ACM Transactions on Networking*, 2015. 19, 27, 29

[49] Y. Cui, X. Wang, H. Wang, G. Pan, and Y. Wang. FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol. In *Proc. of International Conference on Distributed Computing Systems*, 2012. 27, 29

[50] Yong Cui, Lian Wang, Xin Wang, Yisen Wang, Fengyuan Ren, and Shutao Xia. End-to-end coding for TCP. *IEEE Network*, 2016. 17, 27

[51] R. d. O. Schmidt, R. Sadre, N. Melnikov, J. Schonwalder, and A. Pras. Linking network usage patterns to traffic Gaussianity fit. In *Proc. of IFIP Networking*, 2014. 6, 42

[52] R. d. O. Schmidt, R. Sadre, and A. Pras. Gaussian traffic revisited. In *Proc. of IFIP Networking*, 2013. 4, 6, 41

[53] R. d. O. Schmidt, R. Sadre, A. Sperotto, H. van den Berg, and A. Pras. Impact of Packet Sampling on Link Dimensioning. *IEEE Transactions on Network and Service Management*, 2015. 4, 41, 42

[54] R. d. O. Schmidt, H. van den Berg, and A. Pras. Measurement-based network link dimensioning. In *Proc. of IFIP/IEEE IM*, 2015. 6

[55] T. Das and K. M. Sivalingam. TCP improvements for data center networks. In *Proc. of COMSNETS*, 2013. 40

[56] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. of OSDI*, 2004. 2, 18, 19

[57] B. Di, H. Zhang, L. Song, Y. Li, and G. Y. Li. Ultra-Dense LEO: Integrating Terrestrial-Satellite Networks Into 5G and Beyond for Data Offloading. In *IEEE Transactions on Wireless Communications*, 2019. 1

[58] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Ph. Stoecklin. On the 95-Percentile Billing Method. In Sue B. Moon, Renata Teixeira, and Steve Uhlig, editors, *Proc. of PAM*, 2009. 43

[59] A Dixit, P Prakash, Y C Hu, and R R Kompella. On the impact of packet spraying in data center networks. In *Proc. of INFOCOM*, 2013. 3, 13, 19

[60] Allen B Downey. Evidence for Long-tailed Distributions in the Internet. In *Proc. of ACM SIGCOMM Workshop on Internet Measurement*, 2001. 4, 42

[61] B. Efron and R. Tibshirani. An Introduction to the Bootstrap. *Chapman and Hall,New York*, 1993. 46

[62] Paul Emmerich, Maximilian Pudelko, Simon Bauer, and Georg Carle. User Space Network Drivers. In *Proc. of of the Applied Networking Research Workshop*, ANRW '18, 2018. 12

[63] C. Raiciu et al. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proc. of SIGCOMM*, 2011. 3, 19, 20, 29

[64] D. Li et al. Reliable Multicast in data center networks. In *IEEE Transactions on Computers*, 2014. 2, 13

[65] G.Parisis et al. Trevi: Watering Down Storage Hotspots with Cool Fountain Codes. In *Proc. of HotNets*, 2013. v, ix, 14, 33, 48

[66] M. Handley et al. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. of SIGCOMM*, 2017. 3, 5, 6, 12, 13, 14, 19, 23, 25, 29

[67] M.Luby et al. RaptorQ Forward Error Correction Scheme for Object Delivery. *IETF, RFC 6330*, 2011. 5, 32, 36, 38, 92

[68] N.Mohammad et al. DCCast: Efficient Point to Multipoint Transfers Across Datacenters. In *Proc. of HotCloud Workshop, USENIX*, 2017. 13

[69] P. Loiseau et. al. Investigating self-similarity and heavy-tailed distributions on a large-scale experimental facility. *IEEE/ACM Transactions on Networking*, 18(4):1261–1274, 2010. 4

[70] S. Ghemawat et. al. The google file system. In *SOSP*, 2003. 2, 18

[71] Muhammad Zubair Farooqi, Salma Malik Tabassum, Mubashir Husain Rehmani, and Yasir Saleem. A survey on network coding: From traditional wireless networks to emerging cognitive radio networks, 2014. 26

[72] Nathan Farrington and Alexey Andreyev. Facebook's data center network architecture. *IEEE Optical Interconnects*, 2013. 1

[73] Daniel Firestone and et al. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proc. of NSDI, USENIX*, 2018. 11, 12

[74] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, and Neal Cardwell. Reducing Web Latency : the Virtue of Gentle Aggression. In *Proc. of SIGCOMM*, 2013. 2, 17, 27

[75] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1993. 17

[76] R. Fontugne, P. Abry, K. Fukuda, D. Veitch, K. Cho, P. Borgnat, and H. Wendt. Scaling in Internet Traffic: A 14 Year and 3 Day Longitudinal Study, With Multiscale Analyses and Random Projections. *IEEE/ACM Transactions on Networking*, 2017. 42

[77] A Ford, C Raiciu, M Handley, and O Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. *IETF, RFC 6824*, 2013. 20, 21

[78] A. Ford, R. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. *IETF, RFC 6182*, 2011. 20

[79] C Fraleigh, F Tobagi, and C Diot. Provisioning IP backbone networks to support latency sensitive traffic. In *Proc. of IEEE INFOCOM*, 2003. 43

[80] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: Distributed Near-optimal Datacenter Transport over Commodity Network Fabric. In *Proc. of CoNEXT*, 2015. 3, 19, 22, 29

[81] José Luis García-Dorado, José Alberto Hernández, Javier Aracil, Jorge E. López de Vergara, and Sergio Lopez-Buedo. Characterization of the busy-hour traffic of IP networks based on their intrinsic features. *Computer Networks*, 2011. 42

[82] L. Golubchik, S. Khuller, K. Mukherjee, and Y. Yao. To send or not to send: Reducing the cost of data transmission. In *Proc. of IEEE INFOCOM*, 2013. 43

[83] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David a. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network Albert. In *Proc. of SIGCOMM*, 2009. 9, 11

[84] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N.M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. Queues don't matter when you can JUMP them! In *Proc. of NSDI USENIX*, 2015. 3, 19, 24, 29

[85] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: : A High Performance, Server-centric Network Architecture for Modular Data Centers. In *Proc. of SIGCOMM*, 2009. 5, 9, 11, 91

[86] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over Commodity Ethernet at Scale. In *Proc. of SIGCOMM*, 2016. 13

[87] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *Proc. of SIGCOMM*, 2008. 9, 11, 91

[88] Z. Guo, J. Duan, and Y. Yang. On-Line Multicast Scheduling with Bounded Congestion in Fat-Tree Data Center Networks. *IEEE Journal on Selected Areas in Communications*, 2014. 13

[89] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 2008. 14

[90] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. In *IETF RFC 5348*, 2003. 27

[91] Mark Handley. Delay is Not an Option: Low Latency Routing in Space. In *Proc. of HotNets*, 2018. 1

[92] Mark Handley. Keynote Video time 32:45. In *Proc. of the ACM Special Interest Group on Data Communication*, SIGCOMM, 2019. 91

[93] Guanghui He and Jennifer C. Hou. On sampling self-similar Internet traffic. *Computer Networks*, 2006. 42

[94] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. In *IETF, RFC 6582*, 2012. 2, 14

[95] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *Proc. of SIGCOMM*, 2012. 12

[96] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm Status. *IETF, RFC 2992*, 2000. 3, 13, 21

[97] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He. CAPS: Coding-based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center. In *IEEE INFOCOM*, 2018. 19, 27, 29

[98] infinibandta. Infiniband Trade Association. RoCEv2., howpublished = Online; accessed August 2019, note = `https://cw.infinibandta.org/document/dl/7781`, year = 2014,. 3, 11, 13, 26

[99] Manish Jain and Constantinos Dovrolis. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. of SIGCOMM*, New York, NY, USA, 2002. 40

[100] By Jay, Kumar Sundararajan, Devavrat Shah, Muriel Me, Szymon Jakubczak, and Michael Mitzenmacher. Network Coding Meets TCP : Theory and Implementation. *Proc.of the IEEE*, 2011. 26

[101] JGroups. JGroups A Toolkit for Reliable Messaging. `http://www.jgroups.org/overview.html`, 2019. [Online; accessed 19-July-2019]. 2

[102] Changlin Jiang, Dan Li, and Mingwei Xu. LTTP: An LT-code based transport protocol for many-to-one communication in data centers. *IEEE Journal on Selected Areas in Communications*, 2014. 19, 27, 29

[103] Hao Jiang and Constantinos Dovrolis. Why is the Internet Traffic Bursty in Short Time Scales? In *Proc. of SIGMETRICS*, 2005. 42

[104] Glenn Judd and Morgan Stanley. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *Proc. of USENIX*, 2015. 20, 29

[105] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *Proc. of IMC*, 2009. 30

[106] M. Kheirkhah, I. Wakeman, and G. Parisis. Multipath transport and packet spraying for efficient data delivery in data centres. In *Computer Networks*, 2019. 21

[107] Morteza Kheirkhah. MMPTCP: a novel transport protocol for data centre networks. In *PhD Thesis: University of Sussex*, 2016. 21

[108] Morteza Kheirkhah, Ian Wakeman, and George Parisis. Short vs . Long Flows : A Battle That Both Can Win. In *Proc. of SIGCOMM (Poster)*, 2015. 21

[109] Morteza Kheirkhah, Ian Wakeman, and George Parisis. MMPTCP: A multipath transport protocol for data centers. In *Proc. of INFOCOM*, 2016. 3, 5, 19, 21

[110] Jorma Kilpi and Ilkka Norros. Testing the Gaussian Approximation of Aggregate Traffic. In *Proc. of SIGCOMM*, 2002. 42

[111] Denis Kwiatkowski, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 1992. 92

[112] Dave K Kythe and Prem K Kythe. *Algebraic and Stochastic Coding Theory.* Springer, 2012. 32

[113] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter Bulk Transfers with Netstitcher. In *Proc. of ACM SIGCOMM*, 2011. 43

[114] D. Leith, Rn Shorten, and G. McCullagh. Experimental evaluation of Cubic-TCP. *Proc. of PFLDnet*, 2008. 20

[115] W E Leland, M S Taqqu, W Willinger, and D V Wilson. On the self-similar nature of Ethernet traffic (extended version). In *IEEE/ACM Transactions on Networking*, 1994. 42

[116] D. Li, J. Yu, J. Yu, and J. Wu. Exploring efficient and scalable multicast routing in future data center networks. In *Proc. of INFOCOM*, 2011. 2, 13

[117] Xiaozhou Li and Michael J. Freedman. Scaling IP Multicast on Datacenter Topologies. In *Proc. of CoNEXT*, 2013. 2, 13

[118] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *Proc. of USENIX*, 2013. 11

[119] Charles Loboz. Cloud resource usage heavy tailed distributions invalidating traditional capacity planning models. *Grid Comput.*, 2012. 44

[120] P Loiseau, P Goncalves, G Dewaele, P Borgnat, P Abry, and P V B Primet. Investigating Self-Similarity and Heavy-Tailed Distributions on a Large-Scale Experimental Facility. *IEEE/ACM Transactions on Networking*, 2010. 4, 42

[121] Jos?? Lopes and Nuno Neves. Stopping a Rapid Tornado with a Puff. *Proc. of IEEE Symposium on Security and Privacy*, 2014. 32

[122] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen. One more queue is enough: Minimizing flow completion time with explicit priority notification. In *Proc. of INFOCOM*, 2017. 3

[123] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. *IETF, RFC 5053*, 2007. 32, 36

[124] Michael George Luby, Roberto Padovani, Thomas J. Richardson, Lorenz Minder, and Pooja Aggarwal. Liquid Cloud Storage. *CoRR*, 2017. 33

[125] D J C MacKay. Fountain codes. *IEEE Proceedings - Communications*, 2005. ix, 32, 35

[126] David Malone, Ken Duffy, and Christopher King. Some Remarks on Ld Plots for Heavy-tailed Traffic. In *SIGCOMM*, 2007. 4, 42

[127] M. Mandjes and R. van de Meent. Resource dimensioning through buffer sampling. *IEEE/ACM Transactions on Networking*, 17(5), 2009. 4, 41

[128] George Marsaglia, Wai Wan Tsang, and Jingbo Wang. Evaluating Kolmogorov's Distribution. *Journal of Statistical Software*, 2003. 45

[129] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. In *Elsevier Parallel Computing*, 2014. 2

[130] MAWI. Mawi Archive. Online; accessed August 2019, 2018. `http://mawi.wide.ad.jp/`. 41

[131] M.Balakrishnan, T.Marian, K.Birman, H.Weatherspoon, and L.Ganesh. Maelstrom: Transparent error correction for communication between data centers. *IEEE/ACM Transactions on Networking*, 2011. 27

[132] M. McBride and O. Komolafe. Multicast in the Data Center Overview. In *Huawei Arista Networks draft IETF*, 2019. 2, 13

[133] P E McKenney. Stochastic fairness queueing. In *Proc. of INFOCOM*, 1990. 17

[134] R V De Meent, M Mandjes, and A Pras. Gaussian traffic everywhere? In *Proc. of IEEE ICC*, 2006. 4, 6, 41, 43

[135] Mellanox. Mellanox NICs. Online; accessed August, 2019. `http://www.mellanox.com/page/ethernet_cards%/`. 12

[136] Memcached. Memcached. Online; accessed August, 2019. `https://memcached.org/`. 18

[137] Michael Luby. LT codes. *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002. 27, 32, 34

[138] Microsoft. Microsoft Azure. `https://azure.microsoft.com/en-us/blog/cloud-service-fundamentals-telemetry-reporting/`. [Online; accessed 19-July-2019]. 1, 2

[139] Ming Li, A. Lukyanenko, and Yong Cui. Network coding based multipath tcp. In *2012 Proceedings IEEE INFOCOM Workshops*, 2012. 27

[140] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A Receiver-driven Low-latency Transport Protocol Using Network Priorities. In *Proc. of SIGCOMM*, 2018. 3, 5, 19, 22, 24, 29, 40

[141] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *Proc. of INFOCOM*, 2013. 3

[142] Rizwan Mushtaq. Augmented Dickey Fuller Test. In *SSRN*, 2011. 92

[143] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. The Fundamentals of Heavy-tails: Properties, Emergence, and Identification. In *Proc. of SIGMETRICS*, 2013. 4, 42

[144] NetFPGA. NetFPGA. Online; accessed August, 2019. `https://netfpga.org`. 92

[145] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, Amin Vahdat, and Radhika Niranjan Mysore. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *Proc. of SIGCOMM*, 2009. 9, 11, 24

[146] M. Noormohammadpour and C. S. Raghavendra. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys Tutorials*, 2018. 12, 30

[147] Seo Jin Park and John Ousterhout. Exploiting Commutativity For Practical Fast Replication. In *Proc. of NSDI, USENIX*, 2019. 13

[148] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devarat Shah, and Hans Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *Proc. of SIGCOMM*, 2014. 22

[149] PFC. IEEE DCB. 802.1Qbb - Priority-based Flow Control. Online; accessed August, 2019. http://www.ieee802.org/1/pages/802.1bb.html. 13, 26

[150] Telecommunications Policy. Speed isn't everything: A multi-criteria analysis of the broadband consumer experience in the UK. *Telecommunications Policy*, 2018. 1

[151] Diana Andreea Popescu. Latency-driven performance in data centres. In *PhD thesis, University of Cambridge*, 2019. 18

[152] Pawan Prakash, Advait Dixit, Y Charlie Hu, and Ramana Kompella. The TCP Outcast Problem : Exposing Unfairness in Data Center Networks. In *Proc. of USENIX*, 2012. ix, 3, 5, 14, 16

[153] A. Pras, L. Nieuwenhuis, R. van de Meent, and M. Mandjes. Dimensioning network links: a new look at equivalent bandwidth. *IEEE Network*, 2009. 42, 43

[154] P. Pruthi and A. Erramilli. Heavy-tailed on/off source behavior and self-similar traffic. In *Proc. of ICC*, 1995. 4

[155] QCN. IEEE. 802.11Qau. Congestion notification. Online; accessed August, 2019. https://1.ieee802.org/dcb/802-1qau/. 26

[156] QUALCOMM. RaptorQ Technical Overview. *QUALCOMM*, 2010. ix, 37

[157] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. Data Center Networking with Multipath TCP. In *Proc. of SIGCOMM*, 2010. 3, 5, 13, 17, 27, 92

[158] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. Data Center Networking with Multipath TCP. In *Proc. of SIGCOMM*, 2010. 10, 21

[159] D Raiciu, C., Handley, M., Wischik. Coupled congestion control for multipath transport protocols. *Internet Engineering Task Force (IETF), RFC 6356*, 2011. 20, 21

[160] Vamseedhar Reddyvari Raja, Srinivas Shakkottai, Amogh Dhamdhere, and kc claffy. Fair, Flexible and Feasible ISP Billing. *Proc. of SIGMETRICS*, 2014. 4

[161] A. Rajaei, D. Chalmers, I. Wakeman, and G. Parisis. GSAF: Efficient and flexible geocasting for opportunistic networks. In *Proc. of WoWMoM*, 2016. 1

[162] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. *IETF, RFC 3168*. 20

[163] Vaidyanathan Ramaswami, Kaustubh Jain, Rittwik Jana, and Vaneet Aggarwal. Modeling Heavy Tails in Traffic Sources for Network Performance Evaluation. In G. Sai Sundara Krishnan, R. Anitha, R. S. Lekshmi, M. Senthil Kumar, Anthony Bonato, and Manuel Graña, editors, *Computational Intelligence, Cyber Security and Computational Models*, 2014. 42

[164] R.Mittal, V.Lam, N.Dukkipati, E.Blem, H.Wassel, M.Ghobadi, A.Vahdat, Y.Wang, D.Wetherall, and D.Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proc. of SIGCOMM*, 2015. 3, 19, 26, 29

[165] V. Roca, C. Neumann, and D. Furodet. Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes Status. *IETF, RFC 5170*, 2008. 36

[166] Bilel Ben Romdhanne. Large-scale network simulation over heterogeneous computing architecture. In *Telecom ParisTech*, 2013. 40

[167] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proc. of ACM Conference on Special Interest Group on Data Communication*, 2015. 30, 31

[168] Z Sahinoglu and S Tekinay. On multimedia networks: self-similar traffic and network performance. *IEEE Communications Magazine*, 1999. 42

[169] A. Scharf, M., Ford. Multipath TCP (MPTCP) application interface considerations. *IETF, RFC 6897*, 2013. 20

[170] sFlow. OPEN CONFIG. Streaming Telemetry. Online; accessed August. `http://blog.sflow.com/2016/06/streaming-telemetry.html`. 2

[171] sFlow. Data Plane Development Kit (DPDK). Online; accessed August, 2019. `https://www.dpdk.org`. 11, 12, 92

[172] Muhammad Shahbaz, Lalith Suresh, Jennifer Rexford, Nick Feamster, Ori Rottenstreich, and Mukesh Hira. Elmo: Source Routed Multicast for Public Clouds. In *Proc. of SIGCOMM*, 2019. 2, 13, 14

[173] A. Shokrollahi and M. Luby. Raptor codes. *Raptor Codes, Foundations and Trends in Communications and Information Theory, Now Publisher*, 2011. ix, 5, 32, 33, 34, 36, 38

[174] Amin Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 2006. 32, 36

[175] Alexander Shpiner, Eitan Zahavi, Omar Dahley, Aviv Barnea, Rotem Damsker, Gennady Yekelis, Michael Zus, Eitan Kuta, and Dean Baram. RoCE Rocks Without PFC: Detailed Evaluation. In *Proc. of the Workshop on Kernel-Bypass Networks*, 2017. 13

[176] Arjun et al. Singh. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proc. of SIGCOMM*, 2015. ix, 1, 11, 12

[177] Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High Throughput Data Center Topology Design. In *Proc. of NSDI, USENIX*, 2014. 11

[178] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P.Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Proc. of USENIX*, 2012. 5, 11

[179] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks. *Network Working Group*, 2008. 14

[180] Rade Stanojevic, Nikolaos Laoutaris, and Pablo Rodriguez. On Economic Heavy Hitters: Shapley Value Analysis of 95Th-percentile Pricing. In *Proc. of ACM IMC*, 2010. 4, 6, 43

[181] IEEE Std. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems - Redline. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002) - Redline*, 2008. 18

[182] J. K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network Coding Meets TCP: Theory and Implementation. *Proc. of the IEEE*, 2011. 26, 27

[183] Rohit P. Tahiliani, Mohit P. Tahiliani, and K. Chandra Sekaran. TCP Variants for Data Center Networks: A Comparative Study. *International Symposium on Cloud and Services Computing*, 2012. 28

[184] Omesh Tickoo, Vijaynarayanan Subramanian, Shivkumar Kalyanaraman, and K. K. Ramakrishnan. Lt-tcp: End-to-end framework to improve tcp performance over networks with lossy channels. In Hermann de Meer and Nina Bhatti, editors, *Quality of Service – IWQoS 2005*. Springer Berlin Heidelberg, 2005. 27

[185] Francesco Tonolini and Fadel Adib. Networking Across Boundaries: Enabling Wireless Communication Through the Water-air Interface. In *Proc. of SIGCOMM*, 2018. 1

[186] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (D2TCP). In *Proc. of SIGCOMM*, 2012. 19, 28, 29

[187] Hans van den Berg, Michel Mandjes, Remco van de Meent, Aiko Pras, Frank Roijers, and Pieter Venemans. QoS-aware Bandwidth Provisioning for IP Network Links. *Computer Networks*, 2006. 43

[188] A. Varga. OMNET++ Discrete Event Simulation. In *System User Manual*, 2006. 40

[189] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G Andersen, Gregory R Ganger, Garth A Gibson, and Brian Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *Proc. of SIGCOMM*, 2009. 19, 29

[190] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proc. of the European Conference on Computer Systems (EuroSys)*, 2015. 18

[191] Ashish Vulimiri, Oliver Michel, P. Brighten Godfrey, and Scott Shenker. More is Less: Reducing Latency via Redundancy. In *Proc. of HotNets*, 2012. 27

[192] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of USENIX*, 2006. 2

[193] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowstron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *Proc. of SIGCOMM*, 2011. 28

[194] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *Proc. of USENIX*, 2011. 21

[195] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. ICTCP: Incast congestion control for TCP in data-center networks. In *Proc. of CoNEXT*, 2010. 19, 27, 28, 29

[196] H. Xu and B. Li. Cost efficient datacenter selection for cloud services. In *Proc. of IEEE ICCC*, 2012. 43

[197] H. Xu and B. Li. Joint request mapping and response routing for geo-distributed cloud services. In *Proc. of IEEE INFOCOM*, 2013. 43

[198] H. Xu and B. Li. RepFlow: Minimizing flow completion times with replicated flows in data centers. In *Proc. of INFOCOM*, 2014. 3, 27

[199] Xiaowei Yang. Designing traffic profiles for bursty Internet traffic. In *Proc. of IEEE GLOBECOM*, 2002. 6, 42

[200] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proc. of NSDI, USENIX*, 2012. 2

[201] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *Proc. of SIGCOMM*, 2012. 17, 24

[202] C. Jiang Zheng, D. Li, M. Xu, and K. A Coding-based Approach to Mitigate TCP Incast in Data Center Networks. In *International Conference on Distributed Computing Systems Workshops*, 2012. 3, 5

[203] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion Control for Large-Scale RDMA Deployments. In *Proc. of SIGCOMM*, 2015. 3, 13, 19, 26, 29