Mitigating Transmission Control Protocol Incast Congestion Collapse With Adaptive Resolution Round Trip Time Measurement

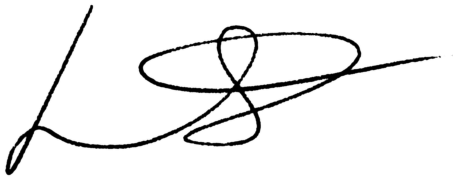
A thesis submitted for the degree of
Doctor of Philosophy

Lawrence A. Stewart
Centre for Advanced Internet Architectures
Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia

November, 2017

I declare that this thesis submitted for the degree of Doctor of Philosophy:

- contains no material which has been accepted for the award to me of any other degree or diploma, except where due reference is made in the text of the examinable outcome.
- to the best of the my knowledge contains no material previously published or written by another person except where due reference is made in the text of the examinable outcome.
- where the work is based on joint research or publications, discloses the relative contributions of the respective workers or authors.



Lawrence A. Stewart	2017-11-30
	Date

In loving memory of

Peter J. Stewart

Greg L. Chesson

Contents

List of Figures

List of Tables

List of Listings

Abstract

The pragmatic use of Transmission Control Protocol (TCP) in high-speed, low-latency clusters and data centres suffers from a throughput collapse pathology associated with a phenomenon known as incast congestion (incast). Incast presents as Round Trip Time (RTT) timescale network congestion, caused by horizontally scaled workloads inducing correlated, high fan-in communication patterns. The timely completion of such workloads is positively linked with business competitive advantage and revenue [1, 2], yet aspects of TCP are poorly tuned for such environments and can variably increase communication latency.

Experimental tools and methodologies for exploring and validating ideas in this space also present a challenge for those without comprehensive access to an operational cluster or data centre – an uncommon luxury outside of large organisations and service providers. To this end, I have designed and implemented the novel CLUster nEtwork Simulation (CLUES) v1.0 hybrid simulation-emulation toolkit to provide such a research platform in support of this thesis. CLUES combines the ns-3 discrete event network simulator, new validated models, Network Simulation Cradle (NSC) framework, and FreeBSD commodity open source operating system network stack.

CLUES is used to investigate incast congestion and the TCP throughput collapse pathology it induces, with a focus on application-layer transaction completion time. The dynamics of a TCP-based, horizontally scaled synthetic workload are studied, showing incast-induced Retransmit Time Outs (RTOs) to be pervasive, and their impact increasingly undesirable as network bandwidth increases and/or latency decreases. The investigation demonstrates that increasing TCP's RTT measurement resolution allows TCP to better adapt to these environments, reducing both the variance and upper bound of transaction completion times.

I have proposed the sender-side, backwards compatible and incrementally deployable Adaptive Resolution RTT mEAsuRement (ARREAR) scheme for TCP, and developed a FreeBSD-based implementation of it which I evaluated using CLUES. ARREAR improves the completion times of RTO-affected transactions by enabling the calculation of sub-millisecond, path-appropriate RTO intervals. The evaluation of ARREAR also reveals that the standard RTO interval calculation specified in RFC 6298 underestimates for such paths, thereby increasing the spurious retransmit rate. An important causal factor in the underestimation is lack of consideration

for serialisation delay bias in RTT measurement, which requires further investigation.

1 Introduction

From humble beginnings in the circuit-switching dominated 1960's, best-effort packet-switched networking and the TCP/IP protocol suite have respectively become the dominant networking paradigm and instantiation thereof. The open and collaborative process adopted to develop royalty-free specifications for the core technologies has contributed to the ubiquitous availability of the protocol suite in network hosts.

The core layering and encapsulation concepts have fostered competition and helped drive commoditisation of hardware and software. Moore's law [3] has held true until very recently, which together with transitor speed improvements have to date delivered exponential growth in digital Integrated Circuit (IC) computational power. This has directly underpinned many advances in networking technology too. For example the Ethernet [4] physical layer has evolved from a few megabits per second in the 1980's to 100 gigabits per second in 2015, and modern Internet Protocol (IP) routers are capable of forwarding more than a terabit (10^{12}) of traffic per second.

The economics resulting from this commoditisation and ubiquitous availability compels us away from vertically integrated custom solutions, and towards generic TCP/IP-based communication substrates. This has inevitably led to numerous "growing pains" from using technologies in environments they were not designed or optimised for. The TCP/IP suite of protocols were originally developed for wide area communication, with typical connection speeds measured in kilobits per second and end-to-end path latencies measured in hundreds of milliseconds.

The nature of interoperable standards and technologies being what they are means that today, devices still only capable of kilobits per second can communicate with those capable of gigabits per second. Similarly, end-to-end paths experienced by the transport layer can be subject to a wide range of capacities and latencies. The process of refining networking technologies to add features, optimise performance and maintain interoperability across such a wide range of variables and use cases is an ongoing challenge.

Cluster-based computing, especially in data centres, represents one such environment where growing pains are still being discovered and worked through. Economics are particularly relevant at data centre scale, and many data centres including the world's largest have embraced the Ethernet/IP network-as-a-substrate world order

with open arms. Here the application of wide-area optimised protocols and the best-effort packet-switched networking paradigm clashes head on with business critical key performance indicators and return on investment.

With the sheer volume of traffic flowing around a data centre, non-trivial wins can be had from minor improvements – shave a few milliseconds off your customer response time here, or a few Central Processing Unit (CPU) cycles off the per-packet processing overhead there and you could well have measurably improved your bottom line. These factors have made the data centre an area of increasing focus for systems and network research and development since the Dot-Com Boom.

The pragmatic use of TCP to provide reliable workload transport within clusters and data centres came about from its required use to serve wide-area Internet services from the same infrastructure. TCP's robustness ensures that it functions correctly in the data centre, but performance is an important consideration for workloads that demand timely completion. For example, client response times for web search and e-commerce are directly linked to competitive advantage and revenue [1, 2]. These demands, coupled with the disparity between high-speed, low-latency data centre network paths and wide-area Internet paths, have led to a body of work on optimising TCP/IP for the data centre with a focus on Flow Completion Time (FCT) as a metric of interest.

A relevant issue that affects FCT and is rooted in the path disparity is a TCP throughput collapse pathology associated with commonly used approaches to distributed computation such as "partition-aggregate" [5]. These computation schemes prescribe how to decompose a problem into a well defined set of sub-problems, map the set onto the available computational resources, and assemble a final result based on what comes back. So called barrier-synchronised workloads [6], which require all of the responses to complete before forward progress can be made, are particularly sensitive to differences between the response FCTs. Barrier-synchronisation requirements are quite common for distributed operations where data consistency and/or completeness is important.

Workloads using such schemes can induce correlated, high fan-in communication patterns which can lead to RTT timescale network congestion – a phenomenon known as incast congestion (or simply incast). When incast drives switch buffers to saturation, the situation is further exacerbated by TCP's standardised congestion control and loss recovery mechanisms being poorly tuned for (multi-) gigabit per second, sub-millisecond latency paths. The timescale of loss events versus TCP's

variable recovery time contributes to a long tail distribution of FCTs, and consequently application layer transaction completion times as well [7]. This variability of service is undesirable from both a user experience and business perspective, with a nett effect that is quantifiable in monetary terms.

A tangential meta issue related to cluster and data centre networking problems like incast is how to research, develop and validate ideas at operational scale. This presents a challenge for those without comprehensive access to an operational cluster or data centre – an uncommon luxury outside of large organisations and service providers. Simulation-based approaches become an appealing option with which to bridge the gap between small scale test bed experiments and the larger experiments required to validate results at operational scale.

A non-trivial challenge with simulation is to ensure that relevant properties of the system are captured in the models and interactions between them so as to make experiments meaningful. Effort can be invested in improving models and the interactions between them, integrating components of real software stacks, and understanding relevant caveats. The improved experimental realism and relevance from these endeavours, combined with the scalability properties of simulation-based approaches, offer a plausible means with which to investigate cluster computing and data centre network issues.

A growing body of literature is devoted to the incast phenomenon, the associated TCP throughput collapse pathology, and various performance metrics. The predominant analytical and experimental focus is on metrics other than application layer transaction completion times, and therefore user experience. Throughput and goodput feature heavily because they are simple to measure and directly relate to the collapse pathology.

Approaches to dealing with incast issues broadly fall into avoidance and/or mitigation categories. Avoidance approaches typically focus on making the network fabric as lossless as possible. Mitigation approaches primarily focus on TCP and its RTO mechanism. They target reducing the duration, prevalence and/or correlation between RTO events, thereby reducing the degree of under utilisation and throughput collapse. Given that many data centres operate under a single administrative domain, a significant portion of work presumes a willingness and ability to have end-hosts work in cooperation with the network.

The review of existing literature reveals a wide range of analysis and approaches for avoiding and/or mitigating incast related issues. Approaches are varied in mech-

anism and the point(s) of insertion in the end-to-end communication path between applications. The predominant analytical and experimental focus on metrics other than application layer transaction completion times, and therefore user experience, misses the forest for the trees. Experimental tools and methodologies for exploring and validating approaches lack focus on the intersection between network protocols, devices and hosts in representative scenarios.

Of the common themes present in the incast literature, there exists a clear appetite for practical though imperfect approaches that target reduction of the FCT distribution's long tail. Keeping the forwarding plane as "dumb" as possible and avoiding application customisation contributes towards practicality. Not being able to utilise perfect knowledge about the end-to-end path therefore leads to a pragmatic desire – adapting the transport protocol to the realities of best-effort Ethernet/IP networks operating at cluster and data centre scale, speeds and latencies.

For TCP, adaptation requires overcoming the impedance mismatch between the intrinsic properties of these networks and TCP's ability to properly detect and react to them. The foundational work of Vasudevan et al. [6] articulates and explores a line of inquiry that is complementary to other work and which this thesis ultimately follows on from – equipping TCP with the ability to measure sub-millisecond end-to-end path RTTs.

This thesis makes contributions towards:

- The understanding of TCP incast congestion in the context of barrier-synchronised workloads.
- Mitigation of the TCP throughput collapse pathology induced by incast congestion.
- TCP RTT measurement, particularly for sub-millisecond latency paths.
- Simulation-based experimental tools and methodologies for cluster and data centre network protocol exploration and validation.

These contributions stem from my pursuit of specific core work items. I have designed and implemented the CLUES hybrid simulation-emulation toolkit to provide a platform for research, development and validatation of cluster computing and data centre network ideas at operational scale. I have conducted a CLUES-based investigation of incast congestion and the TCP throughput collapse pathology it induces,

with a focus on application-layer transaction completion time. I have proposed ARREAR, a sender-side, backwards compatible and incrementally deployable adaptive resolution RTT measurement scheme for TCP, which co-opts the TCP timestamp option to faciliate measurements based on the quality of a sender's local clock(s). I have developed a FreeBSD-based implementation of ARREAR which I evaluated using CLUES. Finally, I have shown that ARREAR improves the completion times of RTO-affected transactions by enabling the calculation of sub-millisecond, path-appropriate RTO intervals.

The remaining chapters present these contributions in the order now outlined. Chapter 2 presents the historical and contemporary background context relevant to this thesis, leading into a detailed review of relevant existing literature in Chapter 3. Chapter 4 introduces the CLUES v1.0 toolkit for network protocol research, development and validation. Chapter 5 presents a CLUES-based investigation of incast congestion, with a focus on application-layer transaction completion times for barrier-synchronised workloads. Chapter 6 proposes the ARREAR scheme for TCP and evaluates a FreeBSD-based implementation of it using CLUES. Chapters 7 and 8 conclude the thesis with a discussion of future work directions and a summary of the key themes, findings and novel contributions.

2 Background

The historical and contemporary context relevant to this thesis' contributions is ordered and related here to help make sense of the starting point for this work.

2.1 TCP/IP: Foundation of the Global Internet

By the 1960's, advances in electronic computing technology and the evident potential of computers prompted discussion about networking the machines to further increase their utility [8, 9]. The Advanced Research Projects Agency NETwork (ARPANET) emerged from this effort in 1969 as the first network of its kind [10], allowing a disparate, geographically dispersed set of machines to connect in an ad hoc manner and multiplex data transmission on an as needed basis across leased lines [11].

The best-effort, connectionless and packet-switched paradigm used by the ARPANET was revolutionary in its simplicity [10]. The equipment complexity and cost could be kept low, redundancy was therefore cheap to build into the network and a lowest common denominator service could be provided. Reliability at the packet level was a non-goal and was the responsibility of network protocols to provide if required. All this was in stark contrast to the highly engineered, circuit-switched telecommunication networks that held the majority mindshare at the time; rigid in their structure and service offerings, providing guaranteed service by way of admission control coupled with complex signaling/scheduling.

What followed the ARPANET's formation was a concerted research, engineering and development effort to establish the discipline of data networking, from which the building blocks for a global Internet emerged. The Internet Protocol (IP) [12] rapidly became *the* convergence layer for packet networks, providing a best-effort datagram delivery service between logically-addressed network hosts [13, 14, 15].

The formation of organisations such as the Internet Engineering Task Force (IETF), Internet Research Task Force (IRTF), Internet Architecture Board (IAB) and Internet SOCiety (ISOC), coupled with work undertaken within the existing International Telecommunication Union (ITU) and Institute of Electrical and Electronics Engineers (IEEE) developed and refined the necessary building blocks for data networking. Architectural aspects like layering/interoperability models [16, 17, 18], key communications (Ethernet [4], ATM [19], X.25 [20], IP [12], UDP [21], TCP [22]), routing (RIP [23], BGP [24]) and application protocols (DNS [25], FTP [26], HTTP [27], TELNET [28]) together formed the standards for data net-

works and network computing.

2.2 TCP/IP Networking Fundamentals

IP networks provide a packet-based, connectionless, statistically multiplexed and unreliable ("best effort") service, which in effect means that the fate of any given packet is uncertain. Packets can be lost, variably delayed, duplicated, corrupted and/or reordered to name a few possibilities. The simplicity and unreliability of the IP layer stems in part from not allocating dedicated resources along the path(s) between communication end points. Instead, the IP layer relies on statistical multiplexing – on-the-fly, localised resource allocation at each individual router (hop). When a packet arrives to be forwarded, either the egress port is free and the packet is forwarded immediately, or the port is busy and the packet is enqueued for eventual transmission.

Statistical multiplexing requires no a priori coordination between the network and end hosts, nor between routers inside the network. This greatly simplifies the addition of new routers and links to expand a network. It does rely on two assumptions though: that the long term average arrival rate is less than the egress port's rate, and the local queues are large enough to absorb likely bursts of correlated packet arrivals. These assumptions break down during periods of transient congestion, or when end hosts sustain an offered load in excess of a path's capacity [29]. Occurrences of either can grow queues to saturation, forcing some packets into the proverbial bit bucket.

Burstiness is particularly troublesome for statistically multiplexed networks, as impulse load builds instantaneous queue, increasing likelihood of packet drops and thereby wasting resources along the path prior to the drop point. The simple act of emitting packets from end hosts at a more even rate (pacing) can have very positive effects on a flow's impact to itself and other flows sharing the path.

When a queue grows beyond one packet, the policy of the scheduling process driving the egress port will determine how the queue is serviced, potentially in an order that differs from arrival. A large body of work exists on what are commonly referred to as Active Queue Management (AQM) schemes, which add intelligence to the process of queue management above that offered by a simple First In First Out (FIFO) scheme.

The choice of egress port is determined by a router's Forwarding Information Base (FIB), which maps destination addresses to the port by which the next hop

along the path should be reached. A router's FIB is determined by its Routing Information Base (RIB), which maps destination addresses to the address of the next hop along the path that a packet should be forwarded to. The FIB can be a mix of explicit forwarding policy programmed by a network administrator, and dynamic routes determined with the aid of a routing protocol.

Routing protocols allow routers to learn about the logical topology of a network by exchanging routing messages with their directly connected peers to share each other's knowledge of the network. By comparing notes with its neighbours, a router can update its FIB so that it knows how to reach networks which may be many degrees of separation away from it.

All devices forming an IP network rely on encapsulation – the practical instantiation of layering at the network protocol level. It allows IP packets to traverse various link technologies and to carry many different upper layer protocol payloads in a programmatically standardised way. Layer-specific information required to accompany data on its journey is associated with the data as wrappers, lowest layer wrapper on the outside.

The localised decision making and statistical multiplexing coupled with short/diurnal/extraordinary timescale variance of offered load means that communication end points experience a channel that is in a constant state of flux. It is the role of transport protocols layered atop IP networks to translate the IP service offering into an offering between communication end points; a practical embodiment of the so called "end-to-end" principle [30].

2.3 Transport Protocols

Transport protocols form an integral role in the layered TCP/IP architecture, providing at minimum an end-to-end (de)multiplexing service. The TCP/IP protocol suite initially specified User Datagram Protocol (UDP) and TCP, but the set of transports has since grown to provide a rich array of service offerings [31]. The changing landscape is forcing us to revisit our assumptions [32, 33] and tweak existing protocols to address limitations [34, 35, 36, 37, 38]. We are also looking beyond these protocols to take advantage of new offerings [39, 40, 41], less constrained by entrenched thinking, assumptions and software Application Programming Interfaces (APIs).

As an example, Stream Control Transmission Protocol (SCTP), with its native multihoming and support for simultaneous data transfer streams [42], offers much

sought after features to application developers and data networks. Efforts are being made to retrofit some of these desired features to TCP as well [43]. Research into transport protocols has therefore intensified in line with Internet growth and the corresponding emergence of issues that affect data transportation in modern networks.

2.3.1 Transport Services

Transports offer consumers a form of contract in terms of the services provided. Given that IP provides an address for data flow, it was obvious from the outset that requiring an address per communication end point would be wasteful and unnecessary. The most basic service provided by all transport protocols is (de)multiplexing, which in the TCP/IP model is achieved by the transport encapsulating a source and destination port number in the packet payload.

The destination port number allows a host to demultiplex received packets into flows, which can then be directed to the correct local end point. Any return communication is addressed to the flow's source address and port number. The combination of protocol, source address, source port, destination address and destination port, commonly referred to as the 5-tuple, constitute the regular means by which packet flows are differentiated.

The IETF TrAnsPort Services (TAPS) working group has categorised and documented the services provided by the set of IETF specified transport and "pseudo-transport" protocols [31]. Only the services which are relevant to TCP will be covered in detail here, and readers are encouraged to review the reference cited for more context.

Reliability refers to a transport's ability to deliver bytes from source to destination end point with some level of assurance as to completeness, correctness and ordering compared to what was sent. It is typically provided by way of some form of sequence number (to determine relative ordering), acknowledgement scheme (to provide feedback to the peer) and checksum over the Protocol Data Unit (PDU) bytes (to help determine if the bytes unintentionally changed along the way). Forward Error Correction (FEC) is a less common transport reliability method, relying on network coding to generate a small stream of redundant bytes to send along with the actual data. A receiver can then reconstruct the intended data in the face of some amount of corruption or loss if some of the redundant bytes arrive instead.

Flow control allows a receiver to inform a sender of an appropriate transmis-

sion rate, typically dynamically during the course of communication. This allows end points without knowledge of each other's capabilities to spontaneously initiate communication without fear of needlessly wasting host and network resources by overwhelming the peer.

Congestion control complements flow control by equiping the transport with a mechanism to avoid overwhelming the network path between end points. Given the almost non-existent direct interaction between end points and the network (schemes like Explicit Congestion Notification (ECN) provide some ability for the network to reliably signal congestion information to end points, though are yet to be widely deployed on the public Internet), congestion control mechanisms to date rely mostly on inference to detect and react to network congestion.

Finally, data boundary orientation refers to the way in which a transport frames byte sequences. As a datagram based transport, UDP only deals in units of individual datagrams. Byte stream transports like TCP present a continuous, unframed stream of bytes without any delineation. The somewhat hybrid message oriented approach offered by SCTP allows for end point defined arbitrary length message framing.

Transport protocols were originally envisaged and designed to operate end-to-end within the TCP/IP architecture, meaning that only end points cared about transport layer encapsulation, semantics and payload. This design philosophy gradually eroded as networks grew in size and became critically important infrastructure. In the name of security (firewalls), performance (Performance Enhancing Proxys (PEPs)), scalability (load balancers) and other causes, networks began to subsume some of the intelligence that was supposed to reside in end points. This gradual intelligence creep within networks has impaired the ability to innovate at the transport layer (e.g., [44]).

2.4 TCP

TCP is a true workhorse, responsible for the vast majority of packets and bytes flowing across the public Internet [45, 46]. Its ability to manufacture a reliable connection between communication end points frees consumers from the need to worry about most details of the underlying network. Together with its algorithms for balancing overall network utility against the individual connection's dynamics and performance, TCP was an instrumental building block for the fledgling Internet's growth.

2.4.1 From Program to Protocol

TCP as we know it today evolved from the "Internet Transmission Control Program" [47, 48] published in 1974. At that time, it combined the network layer functionality and transport layer functionality that we now separately associate with IP and TCP respectively. It was not until 1977 that the architectural revision to functionally separate the network and transport layers was proposed by Jon Postel in Internet Experiment Note (IEN) 2 [49]. His opening discussion remarks bear repeating:

> *"We are screwing up in our design of internet protocols by violating the principle of layering. Specifically we are trying to use TCP to do two things: serve as a host level end to end protocol, and to serve as an internet packaging and routing protocol."*

Jon's potent observation set the agenda for further revisions of TCP [50, 51], which culminated in the publication of the normative reference for TCP (the protocol), Request For Comments (RFC) 793 [22] in 1981. However, RFC 793 did not contain any mention of two goals that would eventually come to be synonymous with TCP: fair share network resource utilisation and network overload protection (congestion control). These came about later in response to an early ARPANET and Internet phenomenon known as congestion collapse [52, 53].

Computing and network resources were costly and at a significant premium in the early days of TCP/IP networks, and therefore they were shared amongst a growing user base. The amount of TCP traffic on the network grew as existing applications switched to using it and new TCP based applications were created. TCP's retransmission mechanism would retransmit any segment for which an ACKnowledgement (ACK) had not been received in a timely fashion, attempting further retransmissions ad infinitum until the ACK was forthcoming or the connection was closed.

As network utilisation grew, the baseline operational state of the network approached the point at which statistical multiplexing assumptions broke down. Router queues would saturate which caused packet loss and increasing latency for packets which were not dropped. The packet loss and increasing latency would trigger legitimate or spurious retransmission timeouts at senders which further increased the load on an already struggling network. The death spiral would eventually grind the network to a halt after queuing induced latency and packet loss cascaded across network routers and the network became saturated with retransmits.

Congestion collapse set the stage for adding fair share resource utilisation and protection against network overload to TCP. The first set of Congestion Control (CC) algorithms to be specified and widely implemented were proposed in 1988 by Van Jacobsen in collaboration with Mike Karels [34]. The work came out of their investigation of the congestion collapse phenomenon that was affecting the network between Lawrence Berkeley National Laboratory (LBL) and UC Berkeley where Jacobsen and Karels respectively worked. Karels was part of the UC Berkeley Computer Systems Research Group (CSRG) that was developing the de facto reference implementation of TCP/IP as part of the 4BSD operating system. He therefore had the ability to rapidly iterate the development and testing of ideas.

Their key insight was that sustained congestion eventually resulted in queue saturation and packet loss, which a TCP sender could infer by way of the duplicate ACKs returning from the peer. Packets are rarely lost in wired networks for any other reason, so the likelihood of false positives was low. In response, the sender could reduce its transmission rate and thereby reduce load on the point of congestion. The proposed changes proved to be effective, incremental and fully contained within the transport layer of end hosts which therefore placed no additional requirements on the network, allowing it to remain as simple as possible. The proposed algorithms were formally mandated for TCP in RFC 1122 [17].

Raj Jain proposed an alternate school of thought in 1989 [54], whereby congestion could be inferred from increasing path latency caused by queue growth as the path is driven in excess of capacity. The more timely and gradual nature of the feedback allows for a more timely and measured response to congestion.

While the idea had significant merit, path delay was a more difficult and noisy signal to measure accurately compared with the more explicit duplicate ACKs used by Jacobsen and Karels. Additionally, the pragmatic loss-based work had already won the deployment race and was seen to be good enough, though with the benefit of hindsight this was arguably an unfortunate fait accompli. Standardising loss-based CC made it difficult to experiment with delay-based CC at scale – as loss-based flows busily drove queues to saturation, delay-based flows unwittingly reacted to the increasing queue occupancy and relinquished their share of network resources.

The development of the world wide web technologies in 1989 [55] and their well suited marriage with TCP's service offering ultimately set the Internet on its trajectory for explosive growth in the early 1990s and beyond.

2.4.2 Operational Overview[1]

TCP's job seems simple – take a stream of bytes from one end point, send them across a network reliably and promptly, then present the same stream of bytes to a peer end point. Doing so whilst adhering to all of its goals turns out to be rather more complex.

TCP uses a positive acknowledgement and retransmission scheme to provide its reliable data transmission service. Each byte to be reliably transmitted is logically mapped into an unsigned 32-bit sequence number space in monotonically increasing order. The lowest sequence number of all bytes that are part of the segment is written into the TCP header sequence number field. A checksum is calculated over any payload and portions of the IP and TCP headers (pseudo header) to include in the TCP header for receiver verification.

The receiving peer verifies the checksum and explicitly acknowledges the data. An ACK segment has the TCP header's ACK flag set and acknowledgement field set to the sequence number of the *next* byte it expects to receive. The acknowledged sequence number is always set relative to cumulatively received bytes i.e. ACKs sent in response to segments that create disorder in the received data stream (known as duplicate ACKs) will all acknowledge the exact same sequence number – that of the first missing byte. Disordered data is sidelined in a reassembly queue until the gaps are filled. Once reassembled, the data can be delivered to the end point and a cumulative acknowledgment covering the previously disordered data returned to the sender. Delaying the delivery of data until it has been reassembled is commonly referred to as head of line blocking, because any disorder in the stream delays delivery of all subsequent data until the disorder is repaired.

TCP maps the SYN (synchronise sequence numbers) and FIN (no more data from sender) control signalling into the sequence space alongside actual payload data. This minimises the protocol's wire footprint by using the acknowledgement scheme to serve a dual purpose and provide reliability for both types of data. It is therefore possible to transmit pseudo bytes, actual end point data, or a mix of both in a single segment, requiring careful differentiation of the relevant bookkeeping.

Being a connection-oriented protocol, TCP will not exchange data until a synchronised initialisation process known as the three-way handshake is completed by

[1]RFC 7414 [56] documents a wide range of standards, extensions, experimental proposals and other informational notes relating to TCP and its evolution. This section covers the key subset of TCP's algorithm and protocol mechanics that are most relevant to incast congestion.

both peers. The communication initiator ("active opener") generates an arbitrary initial sequence number, increments it and includes it in a segment with the SYN (synchronise) flag set. The segment is sent to the address and port number of the intended peer and communication end point. A willing peer ("passive opener") will respond to the unsolicited SYN segment with a SYN segment containing an arbitrary initial sequence number of its own that also ACKs the active opener's SYN. On receipt of the passive opener's SYN/ACK segment, the active opener transitions to the `ESTABLISHED` state and completes the handshake by returning an ACK segment which acknowledges the passive opener's SYN. The passive opener finally transitions to the `ESTABLISHED` state on receipt of the final ACK.

The TCP finite state machine governs the formal lifecycle of a connection at each end point, defining a number of states that represent the opening, established and closing phases. It also specifies the events that trigger, and are triggered by, transitions between them. Events take the form of explicit signalling between peers, and local end point actions such as timers firing or a system call on a socket. For example, the socket `connect` system call will trigger a SYN segment to be sent and the active opener's connection to transition from `CLOSED` to `SYN_SENT`.

Numerous algorithms and timers are also used to overcome various issues inherent in TCP's design/implementation, and to ensure that connections do not end up deadlocked. The subtle intricacies involved in getting all these interactions right are still a constant source of performance issues, bugs and security vulnerabilities in spite of implementations maturing over decades.

TCP's RTO timer mechanism in particular exists as a robust safety net that facilitates recovery from a range of scenarios that would otherwise deadlock a connection. RTOs are often discussed in the context of performance because the intentionally conservative timeout interval causes significant gaps in transmission when relied upon.

For example, transaction based workloads which incur loss(es) near the transmission's tail may not generate sufficient (or any) duplicate ACKs to trigger a fast retransmit. In this scenario, the transaction has no choice but to wait for the retransmit timer to fire and trigger the retransmit of the missing data. The RTO interval directly adds to the transaction completion time, and potentially reduces overall network utilisation.

The conservatism with which the RTO interval is calculated was a deliberate design choice to avoid unnecessary, or spurious, retransmissions from potentially

exacerbating an already congested network. Consecutive RTOs are inferred to indicate increasingly heavy congestion, and a binary exponential backoff scheme [34] is employed to increase the interval between retransmissions.

Many algorithms and mechanisms have been proposed, and to a lesser extent standardised, to reduce the number or frequency of situations in which a RTO is required. Nevertheless, there is no comprehensive set of changes in existence that completely negates the need for them, and so they remain a necessity, to be avoided where possible.

Send Window Control

TCP allows a dynamically variable amount of data to be in flight (sent but unacknowledged) at any given time by way of its send window control mechanism. TCP tracks the edges of the window per RFC 793 variables *snd.max* and *snd.una*. *snd.max* tracks the sequence number of previously unsent data to send next i.e. one above the highest in flight sequence number, and *snd.una* tracks the highest sequence number cumulatively ACKed by the peer. The difference between the edges ($snd.max - snd.una$) represents the current in flight window size.

Returning ACKs for in flight data (i.e. which increment *snd.una*) reduce the in flight window and can be used to trigger the transmission of new data to maintain the same window. This mechanism of triggering the transmission of new data on receipt of an ACK for old in flight data is known as ACK clocking, and serves an important role in spreading the transmission of data to reduce burstiness.

Optimal throughput is achieved when the send window's value for a given path matches the path's Bandwidth Delay Product (BDP) – the amount of data required to drive the path at full utilisation. When equal, the last segment of the window will be transmitted as the ACK for the window's first segment arrives at the sender. BDP is a function of the minimum ("bottleneck") bandwidth along a path and the RTT i.e. the cumulative One Way Delay (OWD) between peers.

A send window that is too small results in lost transmission opportunity while the sender waits for returning ACKs. A send window that is too large drives the path in excess of capacity which will build a standing queue at the bottleneck and eventually saturate it. The challenge is therefore to ensure that the send window promptly reaches, and then tracks, the path's BDP throughout a connection's lifetime. TCP does not have the luxury of knowing a path's BDP, and therefore relies on algorithmic inference and determination to dynamically adapt the send window

over time.

TCP's drive to maximise throughput is tempered by a desire to avoid counter-productive overloading of the receiver and network. TCP's flow control and CC algorithms independently determine a send window limit based on their respective goals, and the sender then caps the effective send window at the minimum of both limits. The interplay between all these send window related factors gives rise to complex micro and macro level intra- and inter-flow dynamic behaviour (dynamics).

Peers advertise a receiving window (*rwnd*) to each other in the header of all segments they send, and it is this information that sender side flow control utilises as its determined send window limit. A receiving peer therefore has overriding control of the sender's send window upper bound given that the sender uses the minimum of the flow control and CC determined limits.

Unlike flow control and the explicit information it uses to determine its limit, CC must attempt to divine relevant information about the path when nothing explicit (the common case) is known. Complicating matters is the expectation that TCP connections will share network capacity with other traffic, and adapt well to all the conditions a modern network path may present (such as latencies fluctuating from sub-millisecond to hundreds of milliseconds, and bandwidths from kilobits to gigabits per second).

Furthermore, a TCP connection is expected to cope in a fair manner (for some notion of fairness) with an unknown but potentially large number of other TCP connections all independently seeking their optimal operating point. Given the intrinsic potential for significant variance over a range of timescales of network load, path characteristics or even the path itself, CC poses a non trivial distributed systems problem. Add to this a desire to optimise for different use cases, and an understanding of the detrimental impact a poorly chosen technique can have on all traffic sharing a network [57, 58]. It is clear why CC remains an active area of research and development to this day.

Congestion Control Algorithms and Dynamics

Standardised CC [34, 59] specifies the algorithms responsible for dynamically determining the send window limit *cwnd*. CC strives to share network bandwidth between like-minded flows and protect the network from being overwhelmed. Three distinct modes of operation with different limit calculations are defined, two of which relate to actively probing a path's capacity, and the other governing recovery from

inferred congestion

Slow start (SS) mode is used to cautiously yet promptly probe the capacity of a path when no meaningful knowledge about the path exists (e.g., on entering the `ESTABLISHED` state or after an extended period of no data transmission). Congestion avoidance (CA) mode is used to maintain reasonable throughput somewhere in the vicinity of the path's capacity over the long term. The *ssthresh* (slow start threshold) variable delimits the *cwnd* size at which the connection transitions from SS to CA mode.

On connection start, *ssthresh* is initialised to the maximum possible window, *cwnd* is set to a finite initial window (IW) value between two and four segments, and the sender operates in SS mode. The IW limits the initial burst of packets transmitted into the network at the sender's full line rate. This is useful because there is no in flight data, ACK clocking is not yet in effect and the connection is typically not flow control limited. In this mode *cwnd* is incremented for every byte of in flight data ACKed, effectively doubling *cwnd* each RTT and resulting in exponential window growth. This continues until either *rwnd* becomes the effective send window limit, or congestion is inferred after receiving three back-to-back duplicate ACKs.

Detecting congestion transitions the sender to fast recovery (FR) mode, which upon entering sets $ssthresh = \frac{cwnd}{2}$, followed by $cwnd = 3 \times SMSS$ and finally performs a fast (compared to waiting for a RTO) retransmit of the presumed lost segment. *cwnd* is presumed to represent the send window at which the path is driven to saturation, thus half that figure would make an appropriate cutover point to switch to CA after exiting FR. Setting *cwnd* to a value that is equivalent to three segments accounts for the three duplicate ACKs that triggered entry to FR mode. Each subsequent duplicate ACK received increases *cwnd* by a Sender Maximum Segment Size (SMSS) to account for another disordered segment arriving at the receiver.

Transmission of new data may continue while waiting for the ACK of the retransmitted segment if the send window allows for it. Receipt of an ACK for *snd.una* triggers the transition out of FR, which sets $cwnd = ssthresh$ and therefore transitions the connection to CA. In the event that the duplicate ACKs are triggered by network induced segment reordering rather than loss, the fast retransmit will have been unnecessary – a so called spurious retransmit.

During CA, *cwnd* linearly increases at the rate of one SMSS per RTT, and like SS,

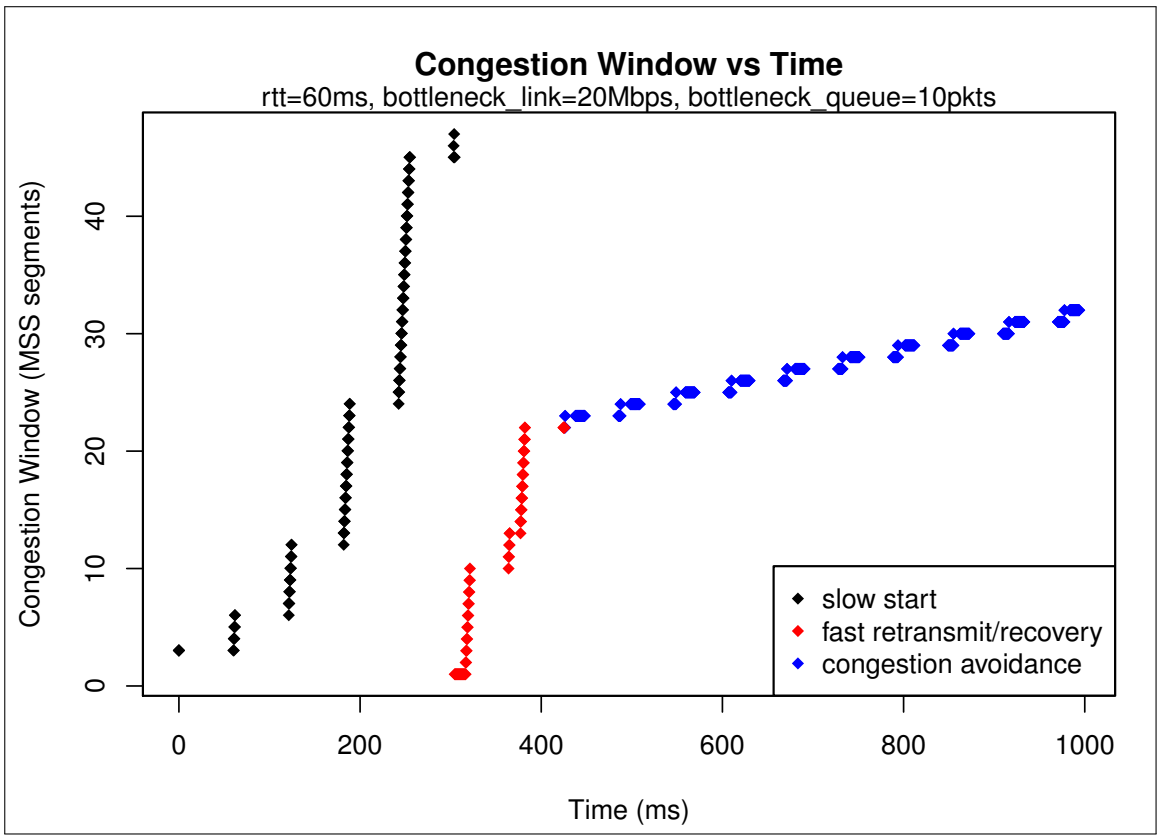


Figure 1: Congestion window versus time for a FreeBSD NewReno TCP flow.

carries on until either *rwnd* becomes the effective send window limit, or congestion is detected. The CA approach to linearly increasing *cwnd* and FR approach to aggressively decreasing *cwnd* is known as Additive Increase Multiplicative Decrease (AIMD).

Figure 1 provides a feel for the window dynamics associated with each of the CC modes[2]. These seemingly simple operational modes and algorithms contribute heavily to the propensity for complex and dynamic intra- and inter-flow behaviour alluded to earlier. A fascinating ecosystem of CC techniques has emerged, each aiming to address a shortcoming of standardised CC [60, 61]. Algorithms most differ in how they grow and shrink *cwnd* in response to network conditions, and are broadly classified as either loss-based or delay-based depending on the path characteristics they use to infer the existence of congestion.

Short-lived flows (e.g., small query-response) are noticeably impacted by TCP's

[2]Experiment utilised a single TCP sender and receiver, communicating via a 20 Mbps, 30 ms one-way delay bottleneck link with a 10 packet queue. The TCP flow utilised a 1448 B MSS and initial congestion window of 3 MSS per RFC 3390.

connection establishment phase and the small IW. Connection establishment takes $1.5RTT$, and an IW of two to four segments forces a multiple-RTT delay on the transfer of application-layer responses. Increasing IW can help short-lived connections, but increases burstiness which is undesirable and can negate the capacity-probing role of SS. Increasing IW to 10 segments has been experimentally proposed [62] based on analysis of Google's vast Hyper Text Transfer Protocol (HTTP) query-response load. TCP Fast Open (TFO) [63], HTTP/2 [64] (and its predecessor SPDY [65]) and Quick UDP Internet Connections (QUIC) [66] all attempt to address this TCP pain point.

By comparison, moderate-to-long-lived flows run into performance problems waiting for *cwnd* to recover after congestion events or idle periods. This is particularly problematic on high BDP paths, as it takes longer to return to the optimal range after each congestion event. RFC 3649 [67] gives standard TCP's average *cwnd* as $w = \frac{1.2}{\sqrt{p}}$ for a steady-state packet drop rate p, making congestion events singularly undesirable. For example, RFC 3649 further observes that to fully utilise a 10 Gbps/100 ms path using 1500 B packets would require an average *cwnd* of 83,333 segments and no more than one congestion event every *100 minutes*. On such a path it would take just over one hour for *cwnd* to regain half the BDP after each congestion event. Many proposals in addition to RFC 3649 attempt to address the slow capacity probing of large BDP paths problem [68, 69, 70, 71, 72].

Apart from the effect on throughput after reducing *cwnd*, loss also manifests as a brief increase in latency at the receiving end point, as the destination waits at least one RTT for the missing segment(s) to be retransmitted. It might seem self-evidently a good thing to minimise packet loss by maximising queue sizes anywhere transient congestion is likely to occur. This more (buffering) is more approach turns out to be very unhelpful in the context of standardised loss-based CC.

Standardised CC relies on regular packet losses as its feedback signal. Without losses, *cwnd* grows until *rwnd* becomes the effective send window limit. If *rwnd* exceeds the path's unloaded BDP (increasingly likely given cheap Random Access Memory (RAM) and TCP window scaling [35]), the excess packets in flight (once $cwnd > BDP$) simply accumulate in the bottleneck queue. The result is a standing queue, which increases the RTT experienced by all traffic sharing the bottleneck without improving the TCP throughput achieved when $cwnd = BDP$. The topic has gained renewed notoriety as "Bufferbloat" – the tendency of operating system and network device vendors to add significant amounts of buffering (BDP) in the network

path without due regard for the consequences [29]. However, the downside of simply adding buffering at congested gateways has been understood and articulated since at least RFC 896 [52]:

> *"Adding additional memory to the gateways will not solve the problem. The more memory added, the longer round-trip times must become before packets are dropped. Thus, the onset of congestion collapse will be delayed..."*

Irrespective of signal timeliness, standardised CC's reliance on loss is problematic for paths which are subject to non-congestive loss (e.g., wireless environments). Unnecessary reductions of the congestion window when a connection is congestion window limited can make it difficult to achieve an appropriate fair share of throughput if the sending rate is frequently below the path's BDP.

It is worth drawing attention to the fact that both flow control and CC rely on imprecise hindsight to determine their send window limit. For active connections, the peer's advertised window used for sender side flow control will probably be stale by the time it has reached the sender OWD later. Similarly, CC relies on inferring the existence of network layer congestion from imprecise indications of the path's state at least one RTT ago. Any increase in delay therefore decreases the responsiveness of TCP's feedback control loop, which in turn increases the length of time TCP may be operating with an inappropriate (too small or large) send window.

2.4.3 RTT and the TCP Control System

RTT fundamentally affects TCP's dynamic behaviour and is a key input to the protocol's control machinery. RTT implicitly affects connections by dictating the feedback latency, which has implications for overall responsiveness and path utilisation per the interaction between BDP and send window. It also explicitly affects connections, by way of measured RTT, as the basis for determining the RTO interval. This also affects responsiveness and path utilisation with flow on effects for various timers and state machine transitions. Although not core to TCP's specification, measured RTT also features as an important control input for many TCP related proposals and schemes (e.g., delay-based CC and fast recovery mechanisms).

At any instant in time, there exists both an intrinsic and dynamic component to a path's RTT. The intrinsic component comprises minimally variant factors inherent to the path itself like signal propagation delays and data serialisation times. The

dynamic component comprises factors inherent to statistically multiplexed systems, like queue occupancy. Measuring the intrinsic and dynamic components of RTT separately is not a consideration for TCP's RTO calculation, which is concerned with selecting a conservative interval based on the current overall path RTT.

There is no practical means by which TCP can know the current end-to-end path's instantaneous RTT. Without an oracle to consult, the RTT has to be measured, and the act of taking a measurement by definition gives you the instantaneous RTT that *was* rather than *is*. A great deal of TCP's specification/implementation complexity and performance challenge stems from the fact that it has to make decisions in the present using stale and/or inferred information, RTT being one such example.

Measuring RTT

Prior to the publication of RFC 1323 [35], TCP stacks would perform RTT measurements in a typically serial manner by pairing the transmission time for an arbitary segment of data with the receive time of that data's corresponding ACK.

The now ubiquitous TCP timestamp option specified in RFC 1323 was conceived as a multi-purpose protocol extension to improve RTT measurement and handling of packet sequencing issues. It specifies the addition of a monotonically increasing 32-bit timestamp value (TSval) to each packet, which the peer reflects back verbatim as an echo reply (TSecr) in the next return ACK packet. The RFC goes on to describe how RTT measurements can be derived from packets which meet certain criteria by calculating the delta between TSecr and the current timestamp clock. This scheme has a number of useful properties:

- The sender does not need to maintain any measurement state which reduces memory consumption and code complexity.
- Other than the monotonicity requirement, the sender's TSval semantics (e.g., rate of increment), do not need to be known to the peer.
- The number of measurements a sender can make scales proportionally with the send window, which helps to minimise undersampling error.

Adapting TCP's control loop to the path would be pretty straight forward if that were the whole story; senders could make accurate RTT measurements by selecting a timestamp clock rate of appropriate resolution, which the rest of TCP's control

logic would ingest and adapt to accordingly. To understand the more complicated reality, let us take a detour via some important historical context.

RFC 793 [22] relied on sequence numbers combined with the concept of Maximum Segment Lifetime (MSL) (RFC 793 arbitrarily assumed a 2 minute worst case) to ensure correctly ordered assembly of the received data stream. This became problematic as the performance of IP networks rapidly improved, because it became plausible for a connection's sequence space to wrap inside of a MSL period. This made it possible for a duplicate segment from a previous window of data to arrive at a receiver, pass the in-window sequence check and corrupt the assembled data stream.

The problem could have been solved a number of ways, but a pragmatic opportunity existed to utilise the freshly minted timestamp option for this in addition to RTT measurement. The result was development and specification of the Protect Against Wrapped Sequence Numbers (PAWS) mechanism.

PAWS is a receiver side mechanism for connections utilising the timestamp option. PAWS extends the set of validity checks applied to incoming segments by comparing a segment's TSval against the cached TSval of the most recently accepted segment (TS.Recent). Only if the former is not less than the latter is the segment eligible to be accepted and have its TSval cached as TS.Recent. The same definition of "less than" (see equation 1, computed in unsigned 32-bit arithmetic) and associated assumptions that relate to TCP sequence numbers also apply to timestamp comparison, ensuring correctly behaved logic even across a field wrap. This definition is also the reason for imposing a monotonicity requirement on timestamp values, which would not otherwise be required if the timestamp option was only used for RTT measurement.

$$s < t \; if \; 0 < (t - s) < 2^{31} \tag{1}$$

The scheme's ability to help a receiver correctly determine segment order depends on a number of factors and assumptions which are discussed in RFC 1323. To recap the relevant issues, the PAWS mechanism provides no help to a receiver if the timestamp increments too slowly such that the sequence number can wrap within the increment interval. It also fails if the timestamp increments too quickly such that the timestamp and sequence number wrap within a similar period of time that is less than the MSL. Both cases would make it possible for a delayed duplicate segment to pass a receiver's segment validity checks and corrupt the data stream.

The PAWS mechansim also introduces a problem of its own. Otherwise valid segments will fail a receiver's PAWS check and be dropped if the delta between TSval and TS.Recent exceeds an absolute value of 2^{31}. This is a property of the modulo unsigned arithmetic used to define "less than" (see equation 1), which can be triggered by a sender's timestamp clock incrementing too quickly relative to the idle period between segments. The failure mode is particularly unfortunate, as all segments sent after the idle period would continue to fail the PAWS check until the delta reaches 2^{32} i.e. wraps back around to 0. Given that TCP has never required a keepalive mechanism to maintain a connection in the `ESTABLISHED` state, imposing a limit on the maximum idle period was considered undesirable.

RFC 1323 discusses all these matters in more detail, and concludes that the timestamp clock frequency should be in the range $[1ms, 1s]$. To mitigate the idle period issue, a mechanism is specified to invalidate TS.Recent. The PAWS check is short circuited if it fails but the elapsed time since TS.Recent was last updated exceeds 24.9 days i.e. the period of time it would take for the fastest suggested clock frequency of 1ms to cause a delta of 2^{31} to be computed.

Many (if not most) popular TCP implementations conform to the RFC, and use a 1ms timestamp clock frequency as well as the TS.Recent invalidation logic[3].

The quality and composition of RTT measurements made using the RFC 1323 prescribed method depends on many factors which receive some discussion in the RFC. Take delayed ACKs for example [73, 17, 59]. Schemes typically send an ACK for every other data segment, and use a timer to trigger an ACK for unpaired segment arrivals. The effective send window limit can therefore influence measurements simply by falling on an odd number of segments boundary.

The RFC suggests that the peer should echo the TSval from the first segment the ACK covers instead of the last, in the hope that the measurement-derived RTO interval will not trigger spurious retransmissions while waiting for the delayed ACK timer to fire. However, this behaviour poses obvious problems for schemes like delay-based congestion control which care about variation of path queuing delay rather than peer contributed delay.

[3] e.g., FreeBSD: `https://svnweb.freebsd.org/base/head/sys/netinet/tcp_input.c?revision=294931&view=markup#l2200`
Linux: `http://lxr.free-electrons.com/source/include/net/tcp.h?v=4.3#L1247`

Consequences of Inaccurate RTT Measurement

On account of RFC 1323 and the PAWS mechanism considerations, current popular TCP stacks do not measure RTTs below 1ms. Whilst generally sufficient for wide-area paths, 1ms is completely inadequate for sub-millisecond latency paths. A fundamental disconnect is created between the TCP feedback latency determined by the true path RTT, and explicitly measured RTT which lacks sufficient resolution to be relevant as a control system input.

A particularly deleterious effect of this disconnect is the inability to calculate an appropriate RTO interval. Transmission opportunity is lost while waiting for the RTO interval to expire. As the performance of a path increases (higher bandwidth and/or lower latency), so too does the negative impact of a TCP RTO event because of the grossly inflated RTO interval.

2.5 Cluster Computing and the Rise of the Data Centre

Cluster computing is a practical manifestion of horizontal scalability – increasing the overal system output/utility by way of increasing the number of workers. The concept is readily observed in nature with eusocial insects such as ants and bees, where colony growth and success is tied to increasing the number of workers to perform required tasks. In contrast, vertical scalability relies on increasing the resources of a single worker to achieve the same outcome.

Whilst not mutually exclusive means of achieveing scale in the computing space, the economics of commoditisation dictate that as required scale increases, so too does the cost effectiveness of the horizontal scaling "axis". A key challenge is engineering the workload and support infrastructure appropriately.

2.5.1 The Pathway to Computing Commoditisation

Due to their expense and size, early computers were only within grasp of governments, big business and well funded academic institutions. There were no standards, and manufacturers were typically vertically integrated selling their own top-to-bottom hardware and software stack. Two important parallel developments which hit a turning point in the 1970s completely changed the landscape: programming languages and personal computing.

Programming languages provide the means by which computer hardware is given instruction. Low level languages are tightly coupled to the specific hardware they

program, requiring the programmer to write code (instructions for the hardware formed using the grammar and semantics of the programming language in use) tailored to that particular hardware. High level languages abstract the details of the underlying hardware, freeing the programmer from the need to tailor all of their code to specific hardware. High level languages therefore require an interpreter – something to turn the abstract code into machine code that the hardware can actually understand. The generic term for such an interpreter is "compiler", which is responsible for translating between a particular programming language and machine code. High level programming languages therefore enabled the possibility to share software between different types of hardware with minimal or no changes required, as long as a suitable compiler existed for the target platform.

The development and specification of the C programming language [74] and its use to rewrite the majority of the fledgling UNIX operating system is of great historical significance. As interest in UNIX grew, C compilers emerged to be able to run it on different hardware. More and more of the operating system was converted from machine specific assembly language into C code which further reduced the effort to port to new platforms [75]. The common abstractions and APIs presented by UNIX were a boon to programmers, allowing them to write general purpose programs within a familiar programming environment that worked across many systems and could therefore be readily shared [76]. This was all pivotal to software commoditisation.

Around the same time, small computers that were suitable and affordable for personal use were gaining popularity amongst electronics hobbyists. They were typically sold as a self assembly kit, and came with a programming manual that described the hardware language used. As hardware and software capabilities improved, Personal Computer (PC) manufacturers turned their eyes to the mass market, focusing their efforts on delivering fully assembled systems that ran useful software. The advent of Graphical User Interfaces (GUIs) significantly broadened the appeal of the PC, and the stage was set for computers to become a common place item in households [77]. The mass market economy of scale precipitated by the personal computing revolution was the other key driver for hardware and software commoditisation.

2.5.2 Data Centres and Incast

A data centre is essentially a centralised location to house back-end systems i.e. computers that people do not need to physically sit in front of. Data centres were

traditionally intra-organisationally focused, housing the systems that provided critical services to an organisation. This was more out of necessity than desire, and the rise of the Internet allowed organisations to obtain many of the information technology services they required from third parties. The growing number of Internet subscribers also created market opportunities for Internet-based companies to deliver goods and services to users as well as organisations, which drove the "dot-com bubble" of the late 1990s [78, 79]. All of the computation and storage required to deliver these goods and services needed to be housed somewhere.

A market developed in parallel with the Internet boom to supply data centre colocation space and hosted resources as a commodity to those who did not build their own. As the Internet continued to grow, so too did the data centre requirements of an ever increasing number of Internet-based companies, and the multi-tenancy data centre market gradually matured as a result. Virtualisation technology has underpinned further evolution of hosted resource data centres into the technology back-end for "the cloud". It allowed data centres to be built more homogenously at greater scale and meet differing tenant requirements using the flexible allocation of resources that virtualisation allows for [80, 79].

Whether leasing resources from a multi-tenancy data centre or building and running custom infrastructure, delivery of ever more complex services at scale has seen cluster computing become synonymous with data centres. Efficiency and performance from every CPU cycle, watt or millisecond of user response time saved furthers competitive advantage.

Being very much geared towards delivering Internet connected services, cluster-based computing inside data centres naturally used TCP/IP for communication. Intra-data centre cluster communication therefore used that which was readily available to it as well. The disparity between high-speed, low-latency data centre network paths and public Internet paths is what ultimately has led to a body of work seeking to optimise TCP/IP for the data centre.

It is with this context in mind that we turn our attention to a particular pain point associated with the use of TCP in the data centre. As discussed in section 2.4.3, TCP relies on accurate RTT measurements as an important control input. Typical intra-data centre path latencies are well below one millisecond [5], and therefore TCP has a significant measurement resolution problem. Being low latency, high bandwidth and typically overprovisioned, data centre networks might be expected to be effectively zero packet loss environments. Unfortunately this is not in fact the

case.

Query-response workloads common to client-server applications, and therefore to data centres, readily lend themselves to a cluster-based, partition-aggregate computation approach [5]. A client issues a query to one of many potential front-end servers. The data from which the response is constructed typically requires computation across a number of distributed services (e.g., [81, 82, 83]) and many back-end machines. An intermediate step is therefore taken to distribute the computation, which the front-end then collates into a client response.

Given that the distribution of work will be approximately equal, the computation time to return a result will also likely be approximately equal. This time correlation of responses combined with the fan-in towards the front-end server can cause an almost instantaneous build up of queues at nearby switch ports. Depending on the nature and number of any losses sustained, TCP may have to resort to one or more RTOs in order to recover.

The order of magnitude or more discrepancy between true RTT, measured RTT, and the stack's RTO_{min} results in the the effective RTO interval being completely inappropriate. The stalls reduce utilisation and goodput, which ultimately increases user response time, bringing undesirable consequences with it [1, 2, 7]. This phenomenon is known as incast, a phrase coined by a clustered storage vendor in the context of their distributed filesystem interacting with their back-end storage node network [84].

2.6 FreeBSD: An Operating System for Cluster Computing

Operating Systems (OSs) interface between the network and applications desiring to communicate over it; turning application data into an appropriate sequence of bits on the wire, and vice versa. The OS network stack acts as a concrete instantiation of the otherwise intangible protocol specifications that define the syntax and semantics of network communication. The IETF early on in its existence recognised the importance of "running code" and independent interoperable implementations to the protocol specification process. Consequently, OS network stacks have been a key vehicle for data networking research and development, with open source stacks in particular providing a readily accessible and free base upon which innovate.

FreeBSD [85] is a mature, open source, UNIX-based operating system that is used in a diverse range of ecosystems, from performance oriented service delivery to embedded consumer electronics devices and indeed, cluster computing as well. It is

derived from the University of California, Berkeley's Berkeley Software Distribution (BSD) version of UNIX, and the BSD network stack was the defacto reference implementation of the TCP/IP protocol suite during development and standardisation within the IETF. Since its inception in 1993, FreeBSD as a project has continued to evolve and innovate its BSD-derived network stack together with other open source communities, industry and academia. FreeBSD and its network stack therefore provide a historically and contemporarily suitable base for cluster and data centre networking research and development.

2.6.1 VIMAGE and VNET

VIMAGE and VNET [86] are FreeBSD virtualisation features available in releases since version 8 and are integral to the CLUES toolkit. VIMAGE refers to the general infrastructure used to virtualise kernel data, on which the VNET virtualised network stack is based. VNET dovetails usefully with FreeBSD's jail [87] feature to offer jails local control of network related configuration. It has both compile-time and run-time requirements in order to function.

Compile-time requirements involve toolchain support and explicit modifications to kernel code. Data to be virtualised are declared as such using the VIMAGE infrastructure [88] and can be assigned a default value if desired. By convention, data are named with a `V_` prefix for clarity. During linking, all virtualised data and their values are gathered together into a "vnet" linker set [89] in the resulting shared object. Each data element is identified by its offset from the start of the block.

At run-time, VNETs are dynamically instantiated using a simple three step operation. First, a new `struct vnet` is allocated and initialised to hold the context for the new VNET. Second, the entire "vnet" linker set block, complete with default values, is copied into a newly allocated block of appropriately sized memory. Finally, the vnet data block's base address is stored in the `vnet_data_mem` member of the previously allocated `struct vnet`.

Kernel threads maintain a pointer to the currently active `struct vnet` context, which subsystems are responsible for setting prior to calling into the network stack. Persistent state associated with various kernel entities (e.g., processes and credentials) determines the correct VNET context to set. When a virtualised piece of data is accessed by a kernel thread executing in the network stack, it is resolved relative to the active VNET's `vnet_data_mem` data block. All changes made to virtualised data therefore only affect the active VNET's data block, while changes

to non virtualised data affect the entire kernel.

VNET trades kernel code complexity against the ability to share the majority of kernel infrastructure between instances. This makes it a comparatively resource efficient, light weight and minimal overhead form of virtualisation.

2.6.2 Development Model

FreeBSD uses a branched software development model, the basic details of which are relevant to parts of this thesis. In FreeBSD parlance, X-STABLE (where X is a major version number) names a conservative development branch from which production-ready X.Y point releases are periodically made. Development occurs on the branch known as [X+1]-CURRENT ("head" branch in FreeBSD's Subversion repository). Development work eventually becomes part of a production release when [X+1]-CURRENT is branched to create [X+1]-STABLE. A backporting process also exists for changes which meet certain criteria to make their way back into existing [X-n,...,X-1,X]-STABLE branches ("stable/X" branches in FreeBSD's Subversion repository) at the discretion of developers.

2.7 Experimental Tools and Methodologies for Cluster-focused Network Protocol Research and Development

Investigating cluster computing and data centre network issues "in the wild" requires comprehensive access to the entire hardware and software stack, as well as a willingness to experiment with production systems and networks. These requirements are not commonly able to be met outside of large service providers who run their own infrastructure. For anyone without such access, validating and extending existing work or developing new ideas is difficult.

Small, topologically representative test beds can be created in a lab at moderate cost to ensure experiments can be performed that are subject to the vagaries of real hardware and software. The costs associated with scaling such experiments to a cluster of more realistic size and/or components quickly becomes prohibitive. Simulation-based approaches become an appealing option with which to bridge the gap between small scale test bed experiments, and the larger experiments required to validate results at operational scale.

A number of benefits and problems are typically associated with simulation-based approaches. Benefits often include faster than real-time experiments, higher

scalability for a given amount of compute and memory resources, arbitary flexibility, fast iteration and ultimate control. These trade-off against the primary problem being a lack of complexity and realism.

Consider a spectrum of "experimental relevance" on which tools and methodologies can be placed. Simplified pure mathematical models exist at the least relevant extreme, with full-scale deployment and verification in the end production environment at the other extreme. Emulation-based approaches fit somewhere in the middle, by blending real hardware and software together with some simulated characteristics (e.g., passing real data flows through a network device that adds artifical delay).

Simulation-based approaches typically occupy the space between mathematical models and emulation-based approaches, but can acquire some of the properties of the latter. Effort can be invested in improving simulation models and the interactions between them, integrating components of real software stacks, and understanding relevant caveats. The improved experimental realism and relevance from these endeavours, combined with the scalability properties of simulation-based approaches, offer a plausible means with which to investigate cluster computing and data centre network issues.

2.7.1 Discrete Event Network Simulators

Discrete event network simulators [90, 91] operate by way of an event loop executing events from a time-ordered event queue until the queue is empty, at which point the simulation is complete and terminates. The simulator engine's virtual clock is stepped to the scheduled time of each event prior to execution, which means complexity (number of events to process) rather than virtual run time (elapsed virtual clock time) determines the amount of real-world time required to complete a simulation. This property of discrete event simulators can translate into a useful benefit of faster than real-time simulations if the complexity is appropriately matched to the underlying host's compute and memory resources.

The ns-2 [92], ns-3 [93, 91] and OMNeT++ [94, 95] discrete event simulators are the most widely represented in cluster and data centre network protocol related literature. They are all open source, community based projects with broadly similar goals, differing primarily in their licence and implementation details.

ns-2 is licenced under a mix of GNU General Public License v2 (GPLv2) and BSD style licences based on its code base's varied lineage. Community development

has gradually quiesced, but it still retains an active user base on account of its familiarity within the research community and rich set of derivative work. ns-2 is based around a C++ core, extensible using C++ and OTcl.

ns-3 is licenced under the GPLv2, and is the logical successor to ns-2 even though its code base started from a "clean slate" without any attempt at backwards compatibility. It is actively developed and used. ns-3 is based around a C++ core, extensible using C++ and Python.

OMNeT++ is licenced such that it is free for non-commercial use, while OMNEST [96] is offered as the commercial use counter part. It is actively developed and used. OMNeT++ is based around a C++ core, extensible using C++. The non-commercial use restriction makes OMNeT++ a less palatable development base where feature differences with ns-3 are not a significant factor.

Riverbed (formerly OPNET) Modeler [97] is a fully commercial network simulator with a long history of development and use. It is based around a C++ core, extensible using Proto-C. Its commercial nature makes it an undesirable base for development and community building.

2.7.2 ns-3

ns-3 is an open source discrete event network simulator which aims to "develop a preferred, open simulation environment for networking research" [98]. It is related to the well known and widely used ns-2 [92] in name only, being a "clean slate" GPLv2 software development effort that is "focused on improving upon the core architecture, software integration, models, and educational components of NS-2" [99].

ns-3 provides a rich set of infrastructure and object models in the form of C++ classes [100], which are combined together as required in a user-defined program and compiled to create a simulation executable. A plethora of models exist, each representing a component or variation thereof which can be used to construct different types of networks, network nodes, applications and other entities.

2.7.3 NSC

NSC defines a generic API that abstracts the plumbing required to interface real-world commodity network stacks with simulated network hosts. It leverages the SCons [101] Python-based build system to provide cross-platform build infrastructure. NSC was initially targeted at ns-2 but has subsequently been incorporated

into ns-3 and OMNeT++. The API was developed in conjunction with ports of the FreeBSD [85], OpenBSD [102], Linux [103] and LWIP [104] network stacks.

NSC stacks are typically compiled as a shared library and dynamically linked into a simulation at run-time using the OS's dynamic linker API. Each simulation can specify which network stack a simulated host will use, and the simulator then instantiates the appropriate stack for each host. ns-3 is given the NSC library path as part of its configuration and stacks are identified by the name of the shared library (e.g., the "libfreebsd-head.so" NSC stack is selected with name "freebsd-head").

The NSC API exchanges fully formed IP packets via the bottom part of the stack, and the standard Berkeley sockets abstraction mediates interaction with the top part of the stack. Porting a candidate stack to NSC therefore typically involves extracting the IP, transport and socket layer code.

3 Data Centre Network Incast and Research Tools

This thesis and its contributions intersect three primary areas of literature: the incast phenomenon, TCP RTT measurement, and experimental tools and methodologies for validating cluster and data centre network protocol research at operational scale.

3.1 Incast

Incast has received a great deal of attention in line with the growing importance of cluster-based workloads and the commodity TCP/IP based networks commonly used as the communications substrate.

A note on terminology – the term incast is used inconsistently across the body of incast related literature, and [105] notes two such discrepancies. Some authors use it to refer to a transport agnostic many-to-one communication pattern. Others use it per its original meaning in [84] to refer to the overall phenomenon of TCP-specific throughput collapse caused by such communication patterns inducing network congestion and associated RTOs. Others differentiate transport throughput collapse from incast congestion i.e. the network-focused effects of highly correlated arrivals at network devices. Context is therefore important when discussing incast related literature

Let us begin with an examination of literature devoting some amount of focus to the investigation, analysis and/or measurement of incast related issues.

3.1.1 Investigation, Analysis and Measurement

The foundational work on incast was based on empirical observation and measurement to understand the relevant parameters as they applied to the clustered storage domain. Nagle et al. coined the "incast" term and were the first to publish details about the phenomenon and some mitigation techniques in the context of their clustered storage product [84]. They identified the causal link between transactional workloads, high fan-in communication, network buffer overruns and TCP's loss recovery mechanisms conspiring to produce throughput collapse.

Expanding on Nagle et al.'s observations, Phanishayee et al. reproduced incast in both ns-2 and a storage cluster test bed to investigate the phenomenon and

evaluate some mitigation strategies in greater depth [106, 105]. They primarily focused on measuring goodput as a function of the number of concurrent servers, thereby independently reproducing and validating the observations of Nagle et al..

Chen et al. followed with a more detailed empirical investigation of TCP incast throughput collapse on a test bed [107]. They explored goodput, number of senders, TCP time sequence plots, congestion window, inter-packet idle time, Smoothed Round Trip Time (SRTT) and RTO count using a synthetic MapReduce-like workload. In later work [108], they presented empirical studies analysing the overhead introduced by incast on Hadoop MapReduce workload components and entire workloads synthesised from third-party production traces. They compared job completion time for regular and 1 ms RTO intervals, and attributed the difference to "incast overhead".

Judd documented operational experience with TCP and various performance problem mitigation measures in a production data centre [109]. A focus of their work was incast mitigation, including reducing RTO_{min} and identifying practical problems with DCTCP deployment that resulted in proposed changes.

Others examined incast within the broader context of data set analysis from operational data centres. Benson et al. analysed data sets collected from a range of data centres [110, 111, 112]. In addition to the empirical analysis, they also extrapolated characteristcs that were used in ns-2 to study various phenomena in fine detail. "Micro-burst losses" were one of the phenomena discussed, which although not directly linked to incast by the authors, is conceptually related to, and likely associated with, incast congestion induced loss. The majority of micro-burst losses were found to be associated with events lasting less than 10 s.

Kandula et al. performed an empirical investigation of a large scale data set collected from an instrumented production cluster [113]. They examined general link congestion and explicitly looked for evidence of TCP incast throughput collapse, noting that it was not directly visible in their socket-level data. They instead comment on the likelihood of incast events having been present based on the frequency with which the requisite conditions were met, and concluded there was very little evidence of collapse occurring in their cluster.

The move towards Ethernet as a converged fabric for data centre connectivity drove research and development of Ethernet fabrics and their interaction with data centre transport/application protocols. Efforts to add congestion control and provide network guarantees for loss-intolerant and/or latency sensitive protocols (e.g.,

from the storage domain) precipitated work with a focus on incast and such converged Ethernet fabrics.

Devkota and Reddy simulated TCP incast scenarios on a Quantised Congestion Notification (QCN) Ethernet and found that QCN is not able to improve the incast throughput collapse pathology on account of an inherent inability to cope with synchronised flows [114]. Anghel et al. investigated the interaction between the New Reno, Vegas and Cubic TCP variants operating over Priority Flow Control (PFC) and QCN Ethernets in both test bed and simulation [115]. Although they did not focus on incast specifically, they found that PFC consistently improved FCTs in the simulation and test bed scenarios examined, and that Vegas was the best perfomer of the three TCP variants if some straight forward changes were made to the implementation.

3.1.2 Modelling

The various empirical investigation and measurement efforts informed a number of modelling efforts.

Chen et al. also developed the first quantitative model for goodput as a function of the number of concurrent senders, transfer size and TCP's RTO_{min}, though the model was acknowledged as being incomplete. Chen et al. later refined and extended their work [108]. They developed and empirically validated a flow rate model to predict goodput before and after the onset of incast in their experiments. They also presented a simple network buffer based model to predict the number of senders bringing about the onset of goodput collapse as a function of TCP slow start congestion window progression and request size. While successful at capturing a number of goodput response features with their model, a number of second order effects were identified which alter the onset of goodput collapse but were too complex to model quantitatively.

Zhang et al. developed an analytical goodput model for TCP NewReno incast throughput collapse and validated it with ns-2 simulation [116]. The model makes numerous simplifying assumptions such as no delayed ACKs, no competing cross traffic, senders are completely synchronised, slow start is ignored, and the number of senders is less than the bottleneck queue size in packets. Feng et al. followed the work of Zhang et al. and developed a broadly similar model, but did include slow start behaviour and validated theirs against test bed experiments [117].

Zheng and Qiao [118], and Chen et al. [119] developed a BDP based model to determine a limit for the number of concurrent senders. Wang et al. developed a BDP and congestion window based model for the onset of incast throughput collapse and validated it using ns-2 [120]. Podlesny and Williamson [121, 122] developed a model that both limits the number of concurrent senders and adds a time-based scheduling dimension. Ke et al. [123] applied the estimation result of [118] to [121] to achieve better throughput results.

Kulkarni and Agrawal developed a throughput based incast model for long lived flows and validated it using ns-2 [124]. They used the well known TCP steady state throughput equation from [125] as a starting point, and extended it to multiple synchronised TCP flows across a single bottleneck link.

Chen et al. developed an interpretive model of incast throughput [126]. They made two key assumptions that they validated using test bed measurements: that full window loss is the primary cause of incast throughput collapse, and the distribution of window sizes amongst flows during any given RTT is normally distributed. Their modelling also assumed the flows are operating in congestion avoidance mode, implying long lived connections without idle periods. A model for the probability of a RTO was also developed as a sub component of the throughput model.

Alizadeh et al. analysed QCN and developed a mathematical model to study the scheme's stability [127]. They also articulated the "Averaging Principle" as a method to improve the stability of congestion control loops in the presence of increasing RTT.

3.1.3 Avoidance and Mitigation

Approaches to dealing with incast issues broadly fall into avoidance and/or mitigation categories. Avoidance approaches typically focus on making the network fabric as lossless as possible, by way of oracle-based or distributed resource scheduling. Transport protocol agnostic approaches typically either modify application communication behaviour, or Ethernet level factors like queue size/management and link level retransmission/congestion control. TCP specific approaches tend to avoid wire format changes, instead opting for transparent packet manipulation or repurposing existing control mechanisms (e.g., modifying the TCP header window field or end-host congestion control algorithm).

Mitigation approaches primarily focus on TCP and its RTO mechanism. They target reducing the duration, prevalence and/or time correlation of RTO events,

thereby reducing the degree of network under utilisation and throughput collapse.

Given that many data centres operate under a single administrative domain, a significant portion of work presumes a willingness and ability to have end-hosts work in cooperation with the network. Approaches that explicitly involve the network in some way are typically better able to increase the overall network utility and performance.

A number of high level surveys covering incast avoidance and mitigation literature have been published, and some provide possible taxonomy points.

Tahiliani et al. surveyed a small subset of TCP modifications and variants for data centre networks and included a discussion of incast [128]. Schemes were classified on where the modification is required (sender, receiver, or switch) and which of the identified data centre network issues (incast, outcast, queue build-up, and sustained queue pressure) are addressed.

Zhang and Ansari surveyed a range of data centre related issues [129]. Included was a comparison of Ethernet congestion notification schemes related to the IEEE 802.1Qau standardisation effort, and a separate discussion of incast mitigation proposals. The Ethernet congestion notification schemes were compared using the following eleven criteria: fairness, feedback control, overhead, rate of convergence to stability, congestion regulation, throughput oscillation, load sensor, link disconnection, fast start, number of rate regulators, and reactive/proactive signalling. No attempt was made to define a taxonomy or set of comparison criteria for the incast proposals discussed.

Pawar and Kulkarni extended the Ethernet congestion notification scheme survey of Zhang and Ansari to include FQCN, but retained the same comparison criteria [130].

Zhang et al. surveyed transport control in data centre networks and included a discussion of TCP incast [131]. Schemes were categorised based on whether they revised TCP, replaced TCP with a different transport, or addressed incast outside of the transport layer.

Ren et al. specifically surveyed TCP incast [132]. They categorised schemes based on the network layer(s) at which they operate, and whether the scheme's modus operandi is to reduce packet loss or improve the time taken to recover from loss. Some schemes at each layer were then compared based on efficiency and cost of deployment.

Rojas-Cessa et al. surveyed schemes for fast transmission of data centre flows

[133]. They produced the most extensive comparison and taxonomical categorisation found in all of the reviewed surveys, and the details are too extensive to summarise here. Despite their Herculean effort, they overlook a significant portion of relevant literature. Their comparison criteria and taxonomical categorisation also fail to establish sufficient detail particularly at the sub-category level (and below). Their efforts provide a sound basis for future categorisation and comparison work.

Sreekumari and Jung surveyed issues, solutions and challenges associated with data centre transport protocols [134]. Proposals were grouped by the issue they addressed (TCP incast, TCP outcast, and latency). Schemes that addressed TCP incast were compared based on end host modification (sender, receiver, or both), switch support requirement, type of congestion control employed (window-based, delay-based, rate-based, or window/recovery-based) and which RTO inducing scenario was avoided/minimised (LHTO, LTTO, and LRTO). Schemes that addressed latency were compared based on deadline awareness, switch hardware modification requirement, ECN requirement, flow size information requirement, and end host TCP modification (sender, receiver, or both).

The taxonomy points and comparison criteria established in the survey literature fall short of capturing the breadth and depth of proposals. Non-survery literature referring to prior work does not apply consistent categorisation. For this literature review, the simple two category classification scheme in Table 1 is used to capture high level groupings.

Containment	Mechanism is fully contained in communicating end points, network devices, or requires cooperation between both
Layers	Which of the network interface, internet, transport or application TCP/IP model layers the mechanism explicitly or semantically changes

Table 1: Incast avoidance/mitigation taxonomy categories.

End point contained, application layer mechanisms

The proposals in this category focus on manipulating application layer logic to control request sizes and/or the number of active sockets at any given time.

Nagle et al. [84] documented an effective modification to a clustered storage product's filesystem striping pattern. The domain-specific modification served to limit the number of cluster nodes concurrently engaged in backend filesystem operations to serve client data requests.

Phanishayee et al. and Krevat et al. [106, 135, 105] extended and generalised the work of Nagle et al. by reproducing incast in a research environment and examining the efficacy of domain nonspecific mechanisms. Explicitly limiting the number of concurrent senders was explored, as was increasing request sizes to reduce the number of concurrent senders and/or increase utilisation during times some flows have incurred timeouts. Staggering data transfers either by modifying the client request or server response characteristics (e.g., intentionally fuzzing the response service time) was used to reduce time correlation between flows. Finally, they suggested but did not investigate the possibility of globally scheduling transfers (e.g., using a token based scheme) as another means of limiting the amount of data able to be concurrently incast towards a receiver.

A collection of works discussed in section 3.1.2 developed simplistic, single switch, model-based approaches to calculating concurrency limits for incast avoidance [118, 119, 121, 122, 123]. These approaches assume homogeneous, single workload environments and require the input of static variables (e.g., switch port buffer size and link speed), thereby limiting their adaptability and generality.

Zhang et al. proposed Optimal Staggering Data Transfers (OSDT) [136], which combines limiting the number, and rate of flows. Yang et al. [137] examined staggering flow start times such that flow end times and TCP window progression have a limiting effect on the amount of incast data. Osada et al. and Kajita et al. [138, 139] examined a more extreme form of staggering with complete and near complete serialisation of flows.

Nishtala et al. discussed the beneficial addition of a request-level, sliding window flow-control mechanism to regulate the number of concurrent requests issued by their UDP-based memcache client [140].

Xu and Li proposed RepFlow [141] and Liu et al. proposed RepNet, a superset middleware scheme consisting of RepFlow and RepSYN [142]. RepNet seeks to minimise FCT for <100 kB flows by replicating them, in the hope that the underlying multipath network and forwarding scheme (e.g., Equal-Cost Multi-Path (ECMP)) may inadvertently find a less congested path for one of the replicated flows. RepFlow replicates flows in their entirety, whereas RepSYN only replicates the TCP hand-

shake and uses the socket which is first to connect for the transaction. The authors did not discuss incast in the context of RepNet, but arguably should have given the potential for replication to induce or exacerbate incast.

End point contained, transport layer mechanisms

The mechanisms in this category broadly fall into implementation tuning, protocol operational enhancements, algorithmic enhancements, or scheduling proposals, with the majority applied specifically to TCP.

Early tuning work carried out by Nagle et al. [84], Phanishayee et al. [106, 105], and Krevat et al. [135] evaluated different TCP implementations (NewReno, SACK), lowering RTO_{min}, lowering the receive socket buffer size to restrict each sender's window, lowering the duplicate ACK threshold, disabling slow start, and restricting the send window. None of the changes in isolation or combination proved entirely effective.

Vasudevan et al. later examined removing RTO_{min} altogether in combination with desynchronising retransmissions and improving TCP's RTT measurement resolution to microsecond granularity [6]. These modifications did not stop incast from occurring, but notably improved aggregate incast goodput by equipping TCP with the ability to compute a path-appropriate (shorter) RTO interval.

Reducing the effect of RTO induced transmission stalls on aggregate goodput was also explored in [143, 144, 145, 146]. Zheng et al. examined removing TCP's binary exponential backoff algorithm to reduce the transmission stall period caused by consecutive retransmit timeouts [143]. Huo and Cao evaluated a hybrid RTO interval calculation scheme, where the Kesselman and Mansour method [147] is used for $N \geq 16$ concurrent senders. Miyayama et al. proposed TCP Fine-grained Timer (TCPFT), which reduced RTO_{min} on the order of hundreds of microseconds and reduced the maximum binary exponential backoff factor to cap the interval for consecutive RTOs [145]. Hafeez et al. proposed and evaluated three different algorithms for randomising RTO intervals to avoid synchronised RTOs triggering synchronised binary exponential backoff [146].

Deng et al. developed the sender-side Minimize the Idle Periods (MIP) scheme, which combined a scheduling mechanism together with removing binary exponential backoff and optimising RTO_{min} for the prevailing conditions [148]. Distributed fair rate control tracks the difference between expected and measured throughput to feedback the required sender rate control to avoid starvation during periods of incast

congestion.

However, the first scheduling-centric proposal was the notionally transport-agnostic Aggregate Transport Control (ATC) scheme from Chesson [149, 150]. By having senders report their instantaneous send backlog with transmitted data (e.g., using a TCP option), the receiver can orchestrate the inbound data flows' windows to minimise bottleneck queue overruns.

Subsequent work also explored receiver-side flow control orchestration schemes. Wu et al. developed ICTCP, which manipulates the advertised window in a Vegas-like manner based on measuring achieved versus expected throughput using a view across all inbound flows [151, 152]. Hwang et al. created the rate-based Incast-Avoidance TCP (IA-TCP) algorithm, which employs ACK pacing for fine-grained rate control combined with ICTCP-like manipulation of advertised windows [153]. Bai et al. proposed Proactive ACK Control (PAC), which uses TCP's ACK-clock mechanism as a means of controlling the amount of, and synchronisation between, in-flight data [154].

Sender-side congestion control algorithms, which traditionally focused on wide-area network congestion issues, followed low latency, high throughput issues like the incast phenomenon into the data centre. Brakmo developed a data centre focused, Vegas-inspired TCP variant named New Vegas (TCP-NV), to be robust in the face of low delay, high bandwidth paths and common hardware offload technologies [155, 156]. Zheng et al. created a data centre focused TCP variant based on FAST TCP [157, 158]. Wang et al. proposed Incast Decrease TCP (IDTCP), a delay-based algorithm that merged the slow start and congestion avoidance window increase algorithms into a single discrete exponential increase function focused on RTT as a congestion signal [159]. Ren et al. developed Stochastic Adjustment TCP (SA-TCP) to desynchronise senders by changing TCP's AIMD congestion window adjustment factors to stochastic values [160]. Zhang and Wen proposed the Ratio Estimation TCP (RETCP) algorithm, which calculates the congestion window as a function of the difference between expected and achieved throughput together with other heuristics over a two RTT horizon [161].

A cluster of work proposed alterations to TCP's loss discovery and recovery strategies to minimise periods of reduced or stalled transmission caused by loss repair and/or RTOs. Kulkarni and Agrawal explored two ideas: a dynamic segment sizing scheme, and a probabilistic, pre-emptive tail retransmission scheme [162, 163, 124]. The former attempts to keep a minimum number of packets in flight regardless of

the send window to increase the likelihood of soliciting sufficient duplicate ACKs to trigger fast retransmit/fast recovery. The latter specially marks and sends pre-emptive tail retransmits that receivers can use to detect tail loss and send four ACKs to trigger fast retransmit/recovery. Tam et al. proposed two sender-side changes: timestamp-assisted retransmission to recover from loss of fast retransmits, and reiterated FINs to detect tail loss at connection termination [164].

Dukkipati et al. proposed the more general sender-side TCP Tail Loss Probe (TLP) scheme for Selective ACKnowledgement (SACK)-enabled connections to recover from tail loss by way of an "ACK overdue" timer tied to the connection's RTT [165]. Cheng and Cardwell specified the sender-side Recent ACK (RACK) loss detection algorithm for TCP SACK-enabled connections, which replaces the duplicate ACK threshold-based fast retransmit trigger with a time based mechanism [166]. Like the complementary TLP, RACK was designed for wide area use, but can be adapted for the data centre.

Jiang et al. proposed the Luby Transform code based Transport Protocol (LTTP) [167, 168, 169]. LTTP combines TCP control channels and UDP data channels utilising TCP Friendly Rate Control (TFRC) [170] for congestion control and LT coding [171] for FEC-based reliability.

Cheng et al. developed ParaVirtualized TCP (PVTCP) to address issues inherent in the operation of virtualised end hosts [172, 173]. PVTCP manipulates TCP's RTT measurement and RTO management machinery to make them more robust against spurious RTOs in the face of hypervisor scheduling induced time skips.

End point contained, network interface layer mechanisms

Zhang et al. explored shrinking the Maximum Transmission Unit (MTU) to increase the number of packets in flight, which in turn may decrease the likelihood of tail loss and associated RTOs [174].

Packet pacing as a rate control mechanism has been employed to augment or serve as a critical component in numerous proposals to reduce congestion amplification caused by sender burstiness (e.g., [175, 176, 177, 178]). Ghobadi and Ganjali explored its effectiveness in data centres [179]. Hardware offload support for pacing large numbers of flows is gradually becoming available [180, 181].

End point contained, cross-layer mechanisms

Parisis et al. proposed a network coding based distributed storage scheme named Trevi [182]. Trevi provides a multicast-based transport service tailored to cluster-based storage systems, employing fountain coding to make it robust in the face of packet reordering and loss.

The remaining proposals in this category effectively propose transport layer mechanisms that rely on assistance from other layers to realise their benefits.

Zhang et al. defined the sender-side Guarantee Important Packets (GIP) scheme [183]. GIP leverages application-supplied request boundary information to reduce the send window at these boundaries in the hope of avoiding an excessive aggregate accumulated congestion window across senders inducing full window loss. GIP also employs packet duplication for each transaction's final packet in the hope of detecting tail loss and triggering fast retransmit/fast recovery.

Hwang et al. proposed the receiver-side, IA-TCP based Deadline and Incast Aware TCP (DIATCP) [184]. DIATCP adds deadline awareness to the receive window allocation and ACK pacing mechanisms proposed by IA-TCP.

Mittal et al. developed TIMELY, a transport-agnostic, delay-gradient congestion control scheme that leverages hardware assisted packet pacing, time stamping and ACK generation capabilities [176]. A rate computation engine ingests the hardware-generated RTT measurements and applies delay-gradient-based AIMD adjustments to determine each flow's appropriate rate relative to a low and high RTT threshold. The hardware rate control engine then appropriately paces the packets to acheive the desired rate.

Network contained, transport layer mechanisms

The mechanisms in this category all propose transparent manipulation by the network of the TCP header's window field to effect incast mitigation.

Abdelmoniem and Bensaou specified the Incast Queue Management (IQM) AQM scheme [185]. Each RTT measurement period, IQM performs a simplistic queue projection and on projected overflow, reduces the window of all flows to one Maximum Segment Size (MSS) until the queue drops below a defined threshold.

Unlike the self-contained IQM, the other proposals used the OpenFlow Software Defined Networking (SDN) standard to manage the flow selection and manipulation machinery independent of where the treatment is applied [186, 187]. Lu and Zhu

developed the congestion-reactive SDN-based TCP (SDTCP), which targets long-lived flows for window reduction during congestion on the presumption that short lived flows are more important [186]. Hwang et al. proposed the Scalable Congestion Control Protocol (SCCP), which proactively apportions a fair share of each port's notional BDP to the port's active flows [187].

Network contained, network interface layer mechanisms

Phanishayee et al. evaluated increasing switch buffer sizes up to 1 MB per port [106, 105].

Shpiner and Keslassy proposed the Hashed Credits Fair (HCF) switch queuing scheme [188]. The flow tuple of frames arriving at a HCF switch are hashed into groups which track the amount of service credits the flow group has remaining. Frames are then enqueued in one of two priority queues, with higher priority given to flow groups that have service credits i.e. have been under-served.

Birke et al. proposed the v-RED scheme for SDN-based network virtualisation overlays in multi-tenant data centres [189]. v-RED performs SDN translation to deliver ECN signals up to the overlay network end points, thereby facilitating incast congestion avoidance.

Wu et al. proposed ECN*, an alternative to DCTCP that leaves end host ECN machinations unchanged [190]. Switches are modified to use instantaneous rather than average queue length as the basis for ECN CE marking, and a marking threshold based on the data centre network's BDP.

Alizadeh et al. developed High-bandwidth Ultra-Low Latency (HULL), a switch-based mechanism that trades network utilisation for predictable low latency [175]. HULL seeks to preempt queue build up by modifying switches to begin ECN CE marking as link utilisation approaches saturation. This is achieved by applying the ECN marking threshold to a "Phantom Queue", which simulates what the queue occupancy would be at some fraction of the true link rate.

Xu et al. proposed the SDN-based Sending In Group (SIG) scheme [191], though insufficient technical detail was communicated to fully understand the proposal. The SDN controller calculates and enforces a limit on the number of concurrent senders allowed to transmit towards a given switch port. Senders are broken up into transmit groups which the switch controls, presumably using a mechanism such as Ethernet pause frames.

Alizadeh et al. proposed CONGA, a distributed, congestion-aware flowlet load

balancing scheme for data centre fabrics [192]. CONGA leverages VXLAN [193] to carry state by encapsulating frames crossing the fabric. Hop-by-hop congestion information between the source and destination leaf switches is accumulated and opportunistically fed back by piggybacking on any reverse direction packets. Switches use the information to maintain a table of congestion on each path towards a leaf and makes forwarding decisions that seek to reduce path congestion. Wang and Xu developed Expeditus, which is conceptually similar to CONGA but flow-based and suited to complex fat-tree topologies [194].

Wang et al. proposed using physical optics to accelerate various data centre communication patterns [195]. Samadi et al. developed the idea further and proposed a SDN controlled hybrid electrical and optical network architecture that can be used to accelerate bulk incast traffic delivery [196].

Lee et al. proposed that SDN network devices hide ECN CE marks from end hosts for certain TCP flows whose window is to be protected [197]. The Differentiated Services Code Point (DSCP) is set by the edge SDN network devices for packets belonging to target flows and used to identify packets which should have ECN CE marks removed

End point and network cooperative, transport layer mechanisms

Jouet and Pezaros proposed a SDN assisted scheme that disseminates dynamically calculated TCP initial window and minimum RTO interval values to hosts during connection establishment [198]. By virtue of being an oracle consulted for all new flow routing decisions (e.g., TCP SYN packets), the SDN controller can perform this role in parallel with route and connection establishment.

Cheng et al. proposed Cutting Paylod (CP) and Precise ACK (PACK) [199]. Network devices are modified to drop the data payload of TCP segments during congestion and forward the truncated packet. Two reserved bits from the TCP header are used for signalling segments that can be cut (set by end host), have been cut (set by network device) or carry a PACK (set by end host). PACKs repurpose the SACK option wire format and are used to inform senders of CP segments that were received so that the data can be retransmitted.

Zhang et al. developed Sharing by Allocating switch Buffer (SAB) [200]. SAB repurposes the window field of the TCP header to instead convey a network imposed limit on the sender's window. Network devices along the path calculate a window limit based on local conditions and update the window field if the local limit is lower

than the current field value. Receiver's reflect the limit in the window field of ACKs instead of advertising their own receive window.

Adesanmi and Mhamdi proposed a similar scheme to SAB named Many-To-One TCP (M21TCP) [201]. A new Maximum Congestion Window (MCW) TCP option is defined and used instead of repurposing the window field of the TCP header.

End point and network cooperative, network interface layer mechanisms

The IEEE 802.3 Ethernet working group defined the PAUSE frame flow control mechanism as part of the 802.3x Ethernet standard [202]. PAUSE frames allow an Ethernet receiver to indiscriminately stop all communication originating from a specified sender for a configurable period of time. Nagle et al. [84] notes Ethernet flow control (PAUSE frames) could be used and Phanishayee et al. [106] explored the idea. Both note that PAUSE frames pose significant issues in networks that do not connect all nodes via a single switch due to trunk port head-of-line blocking.

The IEEE 802.1 Data Center Bridging Task Group defined the Priority Code Point (PCP), QCN, and PFC mechanisms as part of the 802.1Q Ethernet standard [203]. These mechanisms provide building blocks which can be used to construct lossless Ethernet fabrics. PFC provides a finer granularity of control than standard 802.3x PAUSE frames by targeting a particular PCP class. QCN provides a means for network devices to advise senders of graduated congestion at queues and thereby influence the sending rate to curtail the congestion.

A number of proposals were discussed during the 802.1Q congestion notification standardisation process. The initial Backward Congestion Notification (BCN) proposal [204] eventually morphed into QCN which was ultimately adopted. Jiang et al. and So-In et al. proposed Forward Explicit Congestion Notification (FECN) [205] and Enhanced FECN (E-FECN) [206].

Devkota and Reddy explored QCN performance in TCP incast scenarios and proposed changes to QCN's congestion point sampling and reaction point rate adaptation [114].

Kabbani et al. improved the convergence to fairness speed of QCN with Approximately Fair QCN (AF-QCN) [207]. Zhang and Ansari proposed an alternate scheme to improve the fairness of QCN named Fair QCN (FQCN) [208].

Shimonishi et al. proposed Ethernet with Flow Label (EFL) [209], which augments Ethernet to make it function as a reliable, delay-based, congestion-controlled transport in its own right. The scheme is discussed in the context of tunneling

PCI-Express over very short distance Ethernets, but would in theory be capable of carrying regular IP packets.

Rajanna et al. proposed XCo [210, 211], an oracle-based flow scheduling scheme. Local coordinators running on each host react to transmission directives issued by a central controller during periods of congestion, and limit IP-level (i.e. host-to-host) flows in the requested manner.

Vattikonda et al. developed an oracle-based Time Division Multiple Access (TDMA) "fabric manager" to coordinate data centre Ethernet fabrics [212]. The Ethernet PFC [203] mechanism is coopted for signalling slot start and stop times to hosts based on the host requirements communicated to the fabric manager.

Grosvenor et al. combined sender-side rate control together with IEEE 802.1Q Priority Code Points (PCPs) [203] to create QJUMP [178]. By inversely coupling rate with the PCP priority enforced at network devices, applications are able to trade throughput for latency. The scheme also applies intelligence to the rate limiting of each QJUMP level, by admitting packets to the network based on the network's intrinsic transmission rate and forwarding delay, or "network epoch". This imposes a single packet limit on each host's contribution to each priority of network queue during each epoch.

End point and network cooperative, cross-layer mechanisms

Alizadeh et al. devised an ECN-based congestion control modification to TCP named DCTCP [5, 213]. All network switches are configured to ECN mark IP packets when local queue occupancy exceeds a fixed threshold. The sender then uses the *proportion* of marks reflected back to it by the receiver to infer the level of congestion and thereby reduce its congestion window in a proportional manner.

Motivated by some perceived implementation shortcomings with DCTCP, Stewart et al. developed Data Center Congestion Control (DCCC) and Dynamic DCCC for SCTP [214]. DCCC uses the fundamental DCTCP insight of window reduction based on the proportion of ECN marks, but takes advantage of SCTP's richer support for ECN to simplify the end-host implementation. Dynamic DCCC switches from regular ECN backoff to DCCC proportional backoff if the stream's RTT is below 1.1ms.

Das and Sivalingam proposed throughput-seeking incremental modifications to DCTCP and named the result Throughput DCTCP (TDCTCP) [215]. The sender side congestion avoidance window increase was made variable based on α and larger

to more aggressively seek throughput. TDCTCP also adds heuristics related to delayed ACKs to further improve throughput. On the sender side, α is reset if a delayed ACK timeout is incurred to avoid stale congestion information potentially hampering throughput. On the receiver side, the delayed ACK timeout is changed to a dynamic quantity based on the flow's segment arrival rate.

Chen et al. replaced DCTCP's single queue marking threshold at switches with two thresholds to create Double-Threshold DCTCP (DT-DCTCP) [216]. DCTCP was observed to induce queue oscillation in large fan-in scenarios, which was attributed to its single queue marking threshold. DT-DCTCP uses a lower threshold above which to start marking packets, and a higher threshold below which to stop marking packets.

Hwang and Yoo developed the Fine-grained and Scalable TCP (FaST) scheme [217]. FaST combines the DCTCP ECN marking threshold and accounting concepts together with variable segment sizing. Supporting congestion windows below one MSS increases the amount of concurrent fan-in that can be reliably controlled by the scheme.

Xia et al. created the Packet Drop Notification (PDN) scheme [218]. Switches collect the five-tuple, size and TCP sequence number of dropped packets in a PDN record and enqueue records in a side queue associated with the sender's port. PDNs are then piggy backed on the end of Ethernet frames destined for the sender which can use the information to trigger a fast retransmit.

Zats et al. also developed a drop notification scheme, named FastLane [219]. FastLane generates notifications in the data plane as stand alone packets with the same transport header, which are then returned to the sender's IP address. Sender-side modifications to TCP and pFabric to better utilise the drop notifications were also proposed.

Huang and Hu proposed modifications to improve DCTCP's responsiveness to loss, calling the new scheme Enhanced DCTCP (E-DCTCP) [220]. E-DCTCP changes network devices to specially mark and return-to-sender packets that would have been dropped due to congestion. In response to returned packets, senders execute a randomised binary exponential backoff retransmission algorithm to avoid synchronising retransmissions with other potentially affected senders.

Miao et al. took a different approach to FaST with DCTCP+ [177]. Under high fan-in scenarios, the congestion window will reach the minimum non-zero lower bound which may still allow too high a transmission rate to effectively control con-

gestion. DCTCP+ addresses this by employing packet pacing with randomisation to both achieve the required average rate and avoid synchronisation with other senders.

Judd proposed a minor change to DCTCP which they also named DCTCP+ [109]. The change was derived from operational experience difficulties encountered while trying to deploy DCTCP in a production data centre. They showed that DCTCP does not cooexist well with regular TCP and segregated both from one another using DSCP marking and separate network device queues. They also found that not having ECT set on SYN packets resulted in connections often failing to establish in a timely manner. DCTCP+ proposed setting ECT on both SYN and SYN-ACK segments to overcome them being preferentially dropped at network queues when the occupancy was above the ECN marking threshold.

Kato et al. proposed One-sided DCTCP (ODCTCP) and other improvements to DCTCP [222, 221]. ODCTCP calls for minor send and receive side DCTCP changes to avoid performance issues when DCTCP is only available on one side of a connection. Improvements to ECE processing and α initialisation/calculation are also detailed.

Sreekumari et al. based NewDCTCP on DCTCP, but modified the switch ECN marking paradigm and sender-side window calculations [223]. NewDCTCP calls for switches to employ the "mark-front" ECN marking strategy [224]. It also modifies the DCTCP sender-side α and congestion window calculations to be more conservative when more marked than unmarked ACKs are received.

Wilson et al. proposed Deadline-Driven Delivery (D^3) [225], a transport protocol in the same vein as RCP [226] but which is data centre focused and deadline-aware. Applications, hosts and network elements proactively exchange rate-control information and enact rate limiting to optimise the fraction of flows which meet their deadline.

Vamanan et al. further developed the notion of deadline-aware transports with Deadline-Aware Datacentre TCP (D2TCP) [227], but without the deployment barriers inherent with D^3. D2TCP is based on DCTCP, and integrates deadline awareness by weighting the window reduction adjustment based on the application-specified deadline.

Hong et al. proposed the distributed, deadline aware Preemptive Distributed Quick (PDQ) flow scheduling protocol [228]. PDQ is a reliable, connection-oriented transport protocol that explicitly involves the network by way of cooperatively shared fields in the PDQ header. Network devices prioritise flows based on network

policy and the end host supplied PDQ header information. They also feedback rate control information to senders in the PDQ header of ACKs. Applications optionally supply deadline information which influences the prioritisation and allocation of resources at network devices.

Zats et al. created a data centre optimised stack named DeTail [229]. A PFC-based lossless link layer is combined with congestion-aware shortest path packet routing, ECN marking, a simplified TCP resilient to packet reordering, and application specified flow priorities.

Alizadeh et al. developed a minimalist, priority-based data centre fabric named pFabric [230, 231]. End hosts mark packets with an independently and locally determined priority, which network devices use as the sole basis for scheduling and drop decisions. Although compatible with standard transports, a cut down TCP variant is proposed for use with pFabric. It eschews many of the standard TCP heuristics and algorithms in favour of a simplistic rate control mechanism that operates at line rate except when reacting to chronic congestion events.

Chen et al. drew inspiration from deadline aware, FCT optimising proposals such as D2TCP, PDQ and pFabric to create MCP [232]. MCP relies on DCTCP-like ECN marking, deadline awareness and average packet delay measurements to determine appropriate per-flow rates. It is implemented as a sender-side TCP window update function.

Munir et al. proposed the Low Latency Data Center Transport (L^2DCT) FCT minimisation transport protocol [233]. L^2DCT is based on TCP and approximates the Least Attained Service scheduling discipline in order to acheive a reduction in completion times for short flows. L^2DCT modulates the sender congestion window based on estimated flow size and DCTCP-like ECN congestion feedback, with smaller flows receiving larger window allocations.

Munir et al. also proposed Router Assisted Capacity Sharing (RACS) to minimise average FCT [234]. RACS senders initiate rate probing at least once per RTT by attaching a rate request header with flow weight to an outbound packet. Network devices locally compute stateless weighted flow rate assignments based on the number and cumulative weight of flows, and update the rate request hop-by-hop with the lowest path rate. RACS receivers echo the information back to senders to update their sending rate.

Munir et al. deconstructed prior work into three key transport strategies and using them, formulated the Prioritization, Arbitration, and Self-adjusting Endpoints

(PASE) scheme [235]. PASE utilises network abitrators doing coarse timescale inter-flow prioritisation, end hosts independently probing for network capacity, and in-network per-packet prioritisation at sub-RTT timescales. Per-flow arbitration to determine a reference rate and network priority is performed in concert with the path's control plane. The end host transport adjusts the congestion window of its flows per the assigned reference rate, priority and DCTCP-based network congestion feedback. A RTO recovery and packet ordering mechanism is employed to avoid spurious retransmissions when service starvation or priority changes occur.

Bai et al. created the Practical Information-Agnostic flow Scheduling (PIAS) scheme to minimise average FCT [236]. PIAS combines host-based, flow size determined priority marking and DCTCP-based rate control together with priority queuing and DCTCP-based ECN marking at switches. The spurious retransmission avoidance mechanism described in [235] is also utilised.

Ding and Rojas-Cessa proposed the Deadline-Aware Queue (DAQ) flow scheduling scheme [237]. DAQ uses a hop-by-hop credit-based flow control scheme together with priority queuing and weighted round robin scheduling at switches to provide flow size and deadline prioritised service. Network devices assign credits to senders by manipulating the TCP header's advertised window.

Xu et al. proposed Receiver-oriented Datacenter TCP (RDTCP) [238]. RDTCP blends ICTCP-like receiver side advertised window manipulation together with flow priority information and DCTCP-like ECN marking to determine the receive window allocated to each flow.

Tam et al. proposed a switch-assisted flow admission control scheme [164]. TCP packets with the SYN flag set cause the switch to perform a queue projection based on the initial window burst that would proceed connection establishment. If the projection indicates the queue would overflow, the SYN packet is ECN marked to convey the prediction to the end hosts which can then elect to defer connection establishment.

Fang et al. defined Multiple-congestion-points TCP (MTCP) together with a few variations thereof [239]. The scheme repurposes the two bit IP header ECN field as an accumulator, requiring congested devices along the path to increment the field by one up to the maximum value of four. The count, or an average if not exchanging packets one-for-one, is then returned to the sender in the ACK's two bit TCP header ECN field. The congestion window is then adjusted in a manner similar to DCTCP.

Zhang et al. combined the DCTCP ECN marking threshold and accounting

concepts together with delay-based congestion control to create TCP-FITDC [240]. RTT measurements made during a given RTT period are grouped based on the ECE value of the ACK used to make the measurement. The averages of RTTs measured using marked and unmarked packets are then used as the basis for delay-based congestion window adjustments.

Shukla et al. developed TCP Packet Labelling to Alleviate Time-Out (PLATO) [241]. PLATO seeks to increase the number of opportunities for TCP's fast retransmit/fast recovery mechanisms to fully recover from losses, thereby reducing the number of damaging RTO events. The scheme involves end hosts applying DSCP marking to strategically selected packets which network devices must preferentially avoid dropping during congestion. PLATO is able to reduce the likelihood of tail drops and other situations that typically require a RTO to recover from.

Huang et al. proposed the Packet Slicing (PS) scheme [242]. PS relies on the network dynamically calculating the optimal packet size based on fan-in and throughput. The packet size is then communicated to senders using Internet Control Message Protocol (ICMP) "Destination Unreachable" messages as defined for Path MTU Discovery (PMTUD) [243].

Guo et al. explored in-network aggregation schemes that utilise intermediary hosts and/or network devices to reduce fan-in at the ultimate end host [244].

3.2 TCP RTT Measurement Literature

A small body of work exploring modifications to TCP's RTT measurement machinery has been undertaken post RFC 1323 and the introduction of the TCP timestamp option [35].

Vasudevan et al. proposed changing the defacto standard 1ms timestamp option tick granularity on the sender-side to microseconds to alleviate TCP incast congestion collapse [6]. Increasing the resolution of the timestamp clock and using it as the source for outgoing TSval field values allows microsecond granularity RTT measurements to be made. The RTO interval derived from the RTT measurements therefore directly benefits from the better resolution if RTO_{min} is set appropriately.

Scheffenegger et al. proposed a scheme for negotiating a set of extended timestamp option based capabilities [245], and a complementary time interval encoding scheme [246]. The negotiation scheme proposes exchanging signalling during the TCP handshake to agree on timestamp option field semantics for the duration of the connection. A 16-bit timestamp clock interval field is included in the signalling,

which uses the interval encoding scheme from [246] to convey intervals from approximately 16s down to 3.64ps. By exchanging this information, the peers (and devices along the path) can interpret each other's TSval field values and use them for enhanced measurement, algorithmic and other purposes.

Hayes developed the sender-side Enhanced RTT (ERTT) FreeBSD kernel-based TCP RTT measurement scheme [247]. ERTT was developed primarily in support of delay-based congestion control, which requires more measurement accuracy than the stack's SRTT estimate provides. ERTT maintains a list of metadata associated with in-flight data, which it pairs with returning ACKs to produce raw, unsmoothed millisecond resolution RTT measurements.

Mittal et al. leveraged hardware assisted time stamping in their TIMELY scheme to achieve microsecond granularity RTT measurements [176]. The scheme is notionally transport agnostic, but the reality is that the Network Interface Card (NIC) has to understand the transport protocol semantics in order to perform measurements. These measurements could be passed up to TCP for computing SRTT, RTTVAR and the RTO interval, but the scheme intentionally excludes end host processing delay and does not work for bidirectional data transfer. Whether these issues would be overly problematic for RTO interval calculation are not immediately obvious and would require detailed investigation.

3.3 Experimental Tools and Methodologies for Network Protocol Validation at Scale Literature

The ns-2 [92], ns-3 [93, 91], OMNeT++ [94, 95], OMNEST [96] and Riverbed (formerly OPNET) Modeler [97] discrete event network simulators discussed in 2.7 all provide idealised models of key network protocols that are modifiable. These simulators scale based on simulation complexity and the available compute/memory resources.

Jansen developed the Network Simulation Cradle (NSC) to allow real-world commodity network stacks to be integrated into ns-2, and was later ported to ns-3 and OMNeT++ as well [248]. NSC facilitates transport protocol research using real-world code bases at simulation scale by rerouting transport-layer operations via the NSC-abstracted network stack. The "globaliser" mechanism used to isolate per-host network stack data puts downward pressure on the maximum practical number of NSC host nodes in a simulation. The design of the stack ports distributed with NSC

also makes it impractical to maintain them in relation to the upstream code bases which are a constantly moving target.

Lacage developed the Direct Code Execution (DCE) framework for ns-3, which integrates real-world POSIX applications and the Linux network stack with simulated hosts [249, 250]. DCE effectively replaces everything above the simulated network interface of DCE host nodes with real-world code. Tazaki et al. contributed the DCE Cradle, which wraps the DCE network stack in such a way that internal ns-3 application models can use it [251]. This offers a choice between real and modelled application workloads, with the former trading scalability for realism compared to vice versa for the latter.

Kliazovich et al. developed the ns-2 based GreenCloud simulator [252]. GreenCloud is focused on energy consumption for cloud computing data centres rather than network and protocol specific issues. The similarly focused but OMNeT++ based CloudNetSim++ was developed by Malik et al. and relies on the existing OMNeT++ code base for network and protocol level functionality [253].

Birke et al. described modifications to the Venus simulation environment [254] to create a suitable base for their data centre focused research [189]. They modelled a 10 Gbps commodity lossy Ethernet fabric for the network and ported the FreeBSD 9 TCP stack into Venus to provide realistic TCP behaviour. They developed a trace-based synthetic partition/aggregate application workload generator which ran across their sub-hundred node topology.

Zec and Mikuc developed the FreeBSD-based Integrated Multiprotocol Network Emulator/Simulator (IMUNES) [86, 255]. Unlike discrete event simulation approaches, IMUNES uses a light-weight operating system virtualisation approach to create virtual nodes that can be arbitrarily linked together. Nodes therefore run real-world code, communicate in real-time using fully formed connections, and links can be manipulated to emulate various characteristics such as delay. Puljiz and Mikuc further increased the scalability of IMUNES by extending it to run distributed across a cluster [256]. Scalability on the order of thousands of nodes was documented.

Hibler et al. described the FreeBSD-based Emulab open-access virtualised network testbed [257]. Emulab's fundamental architecture is very similar to the enhanced IMUNES described by Puljiz and Mikuc, but its open-access goal and distributed topology from the outset took its feature set in a somewhat different direction to that of IMUNES. Scalability on the order of one thousand nodes was

documented.

Lantz et al. developed Mininet to facilitate rapid prototyping and easy sharing of ideas for large networks on a "single laptop" [sic] (i.e. single resource constrained machine) [258]. Like IMUNES and Emulab, Mininet leverages light weight virtualisation techniques found in Linux to create virtual Ethernet-linked hosts to create arbitrary, functional networks. It has a particular focus on SDN-based networks, and scalability up to the thousand-host mark was documented.

Tan et al. developed the Field Programmable Gate Array (FPGA)-based Datacenter-In-A-Box at LOw cost (DIABLO) warehouse-scale network simulator [259]. DIABLO can comfortably scale to tens of thousands of nodes by using FPGAs to accelerate the simulation of parameterised abstract models similar to those found in discrete event simulators.

3.4 Research Directions

The review of existing literature reveals a wide range of analysis and approaches for avoiding and/or mitigating incast related issues. Approaches are varied in mechanism and the point(s) of insertion in the end-to-end communication path between applications. The predominant analytical and experimental focus on metrics other than application layer transaction completion times, and therefore user experience, misses the forest for the trees. Experimental tools and methodologies for exploring and validating approaches lack focus on the intersection between network protocols, devices and hosts in representative scenarios.

Of the common themes present in the incast literature, there exists a clear appetite for practical though imperfect approaches that target tail reduction of FCT distributions. Keeping the forwarding plane as "dumb" as possible and avoiding application customisation contributes towards practicality. Not being able to utilise perfect knowledge about the end-to-end path therefore leads to a pragmatic desire – adapting the transport protocol to the realities of best-effort Ethernet/IP networks operating at cluster and data centre scale, speeds and latencies.

For TCP, adaptation requires overcoming the impedance mismatch between the intrinsic properties of these networks and TCP's ability to properly detect and react to them. The foundational work of Vasudevan et al. articulates and explores a line of inquiry that is complementary to other work and which this thesis ultimately follows on from – equipping TCP with the ability to measure sub-millisecond end-to-end path RTTs.

4 CLUES: A Cluster Network Simulation Toolkit

Research and development of cluster-focused networking ideas presents a challenge for those without comprehensive access to an operational cluster or data centre – an uncommon luxury outside of large organisations and service providers. Simulation-based approaches become an appealing option with which to bridge the gap between small scale test bed experiments and the larger experiments required to validate results at operational scale.

A non-trivial challenge with simulation is to ensure that relevant properties of the system are captured in the models and interactions between them so as to make experiments meaningful. Significant effort has been invested in open source network simulator projects to develop sufficiently accurate models and provide mechanisms to run production code in simulated experiments. These efforts improve the level of realism and therefore relevance imparted to experiments and results. The CLUES toolkit leverages this prior work and makes additional contributions which combine to form a simulation environment for cluster computing and data centre network research.

4.1 Architectural Overview of CLUES v1.0

The CLUES toolkit combines a discrete event network simulator, commodity open source operating system network stack and new validated models. The benefits and problems typically associated with simulation that were discussed in 2.7 also apply to CLUES. The initial focus was incast congestion in support of the work required to complete this thesis. However, the simulation environment is not specifically tailored to this issue and can be extended as required to facilitate other research topics.

A core goal of CLUES is to support the ability to develop ideas against the production code base which can then be run without modification in the production validation environment. Using the same code base simplifies calibration of the toolkit and increases confidence that the calibrated toolkit captures sufficient realism to ensure the results of simulated experiments are an accurate proxy for testbed results. The network stack is the production code base of interest for transport protocol development, and CLUES utilises the NSC framework to allow commodity network stacks to be ported to the simulator and drive the machinations of simulated host networking.

Those interested in applications and overall system behaviour benefit from the bug-for-bug compatible protocol implementations present in the network operating systems running on production systems. Those interested in protocols gain a useful tool for iterative prototyping and development – arbitrary network topologies and characteristics can be trivially investigated, and code that normally runs in privileged kernel mode where faults manifest as system crashes are no longer disruptive to development. All users benefit from simplified calibration of the simulation and testbed environments on account of results being more closely comparable.

However, CLUES is subject to a number of shortcomings with respect to the realism of experimental results. The core simulator engine runs single threaded and therefore a whole range of bugs and issues related to concurrency are not present in simulated experiments. Specific hardware quirks and hardware/software interactions are not captured in the models and NSC network stack ports (e.g., multiqueue network adapters). Network nodes such as Ethernet switches and IP routers do not run with or attempt to simulate bug-for-bug compatible hardware and software.

Despite these shortcomings, useful results can be obtained if experiments are designed and results interpreted with an understanding of the implications of various implementation details and caveats.

The primary CLUES building blocks are the ns-3 [93, 91] discrete event network simulator, a new ns-3 Virtual Output Queuing (VOQ) Ethernet switch model, a new ns-3 query-response application model for incast-specific synthetic workload generation, the NSC framework [248] for external network stack integration, and a "clean slate" NSC port of the FreeBSD network stack from the FreeBSD "head" Subversion repository development branch.

4.1.1 ns-3

CLUES v1.0 leverages the ns-3 development branch[4]. Additional work from various upstream sources which adds full-duplex operation to the CsmaNetDevice model[5] and nanosecond resolution timestamp granularity to generated pcap trace files[6] was also integrated. CLUES leverages many of ns-3's existing models (some with modifications), and introduces new models specific to cluster and data centre based experimental simulation.

[4]Upstream revision `http://code.nsnam.org/ns-3-dev/rev/ea99cd3b169c`
[5]`http://code.nsnam.org/tomh/ns-3-dev-csmafd`
[6]`http://code.nsnam.org/ns-3-dev/rev/471023dc0c7e`

Investigating incast in its pure form requires highly correlated network traffic originating from multiple sources destined for a common end point. The effects of incast congestion are felt more strongly by applications that require all incoming data be received before a transaction can be completed (also known as a "barrier-synchronised" workload). A query-response client-server application model was created to act as a sequential barrier-synchronised workload generator for incast experiments.

Queuing semantics, forwarding delays and even memory management schemes also influence the dynamic behaviour associated with incast. A new VOQ Ethernet switch model was created with relevant fundamental characteristics roughly comparable to those of commodity data centre switches.

The modified and new models were combined into an incast-specific simulation executable with which to conduct experiments.

4.1.2 NSC

CLUES v1.0 leverages the NSC v0.5.3 source code. Although the stack ports that ship with NSC are outdated and of limited relevance now, the idea of sharing the same network stack between simulated and hardware hosts retains significant merit. For pragmatic reasons related to licencing, my expertise and the VNET virtualised network stack feature, effort was directed towards modernising the FreeBSD port.

The way NSC is integrated with ns-3 is convoluted and far from ideal. The NSC stack and API is wrapped in C++ classes that slot into the ns-3 transport and socket layers, sandwiched between ns-3's IP layer and higher level socket abstraction. The fully formed IP packets required by the bottom part of the NSC stack have to be manipulated to add or remove the IP header for packets going to or coming from the stack. The ns-3 NSC socket wrapper abstraction is not properly integrated with the underlying NSC stack, containing class data and methods which are disconnected from the actual state maintained in the NSC stack's socket object. Event signalling between the NSC stack and ns-3 is unnecessarily expensive and not well aligned with the needs of modern OS stacks.

Despite these and other shortcomings, the integration works sufficiently well if care is taken to understand the implications of the various implementation caveats. Some improvements detailed in 4.4 were made in support of this thesis, but much future work to further improve the integration is possible.

4.1.3 FreeBSD and VNET

A "clean slate" port based on FreeBSD-CURRENT[7] was completed after studying the existing FreeBSD 5.3 based NSC port in detail. Basing the port on FreeBSD-CURRENT allows work to be easily kept up to date with upstream FreeBSD development. It also makes creating ports for STABLE and RELEASE branches trivial. Work targeted for inclusion in upstream FreeBSD can therefore be developed against the Subversion "head" branch as required for merging upstream. It can also be tested against STABLE and RELEASE stack ports to obtain results with the stack used in production.

4.2 A Barrier-Synchronised Workload Generator for CLUES

The new CLUES ns-3 QueryResponseApplication model implements a TCP-based barrier-synchronised synthetic workload. Every ns-3 host the application is installed on can function as both a querier and responder simultaneously. All hosts are passive responders by default, listening on a configurable port for unsolicited queries to respond to. Queriers are implicitly configured when a QueryResponseApplication instance's `AddResponder()` method is invoked to add the address of a responder to query during the experiment. The model offers the configuration options detailed in Table 2.

ListenSockAddr	Address/port to bind the listen socket to
NumQueries	Number of serialised back-to-back transactions to complete
Protocol	ns-3 TypeId of the protocol socket factory to use for socket creation
QuerySize	A RandomVariableStream configured to return the desired distribution of query sizes
ResponseSize	A RandomVariableStream configured to return the desired distribution of response sizes

Table 2: ns-3 QueryResponseApplication model configuration options (inherited options not listed).

[7]Last synced with upstream's "head" branch at revision 287534 dated 2015-09-07.

At the application's configured virtual start time during the simulation:

- The listen socket is created, bound and set to `LISTEN` state, thus configuring the host as a functional responder.
- For each responder address added using the instance's `AddResponder()` method, a new socket is created, bound and connection initiated to the responder peer.
- After all responder peer sockets have successfully connected, the instance will commence the configured number of query-response transactions in a serialised manner.

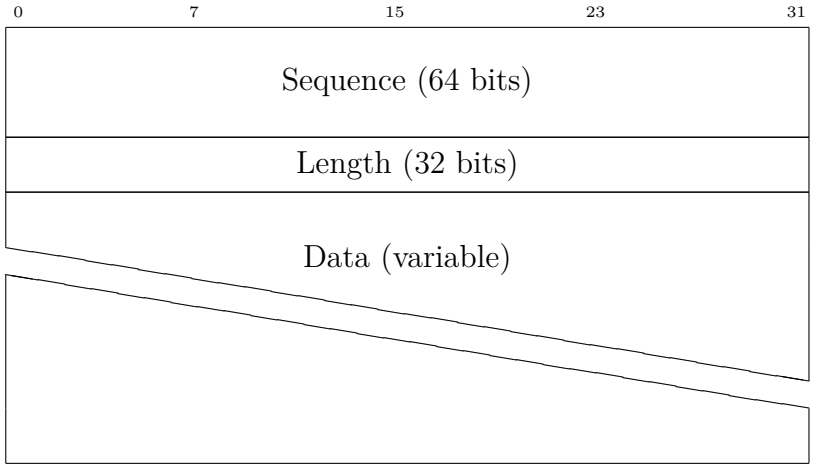


Figure 2: QueryResponseApplication query and response wire format.

Transactions consist of an identical fixed length query being sent to each configured responder, and each responding in turn with a fixed length response. The wire format is documented in Figure 2. The query payload consists of an unsigned 64-bit monotonically increasing sequence number, unsigned 32-bit length in bytes which includes the size of the sequence and length fields, and zeroes as the data. The response payload similarly consists of the unsigned 64-bit sequence number of the query being responded to, an unsigned 32-bit data length, and zeroes as the remainder. The querier tracks the difference between total expected and received response bytes from each responder, and only when all responses have completed in full is the next query issued. Figure 3 details the interactions between a querier and two responders at the TCP flow level with a configured virtual start time of 100 ms. The experiment utilised a 500 B query length and 2 responders returning a 10 kB cumulative response over a 10 Gbps network.

```
|Time       | 10.0.0.1                              | 10.0.0.3          |
|           |                  | 10.0.0.2           |                   |
|0.100000000|         SYN      |                    |                   |Seq = 2832617061
|           |(62086)  ------------------>  (4377)   |                   |
|0.100000072|         SYN      |                    |                   |Seq = 2950880129
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100008785|         SYN, ACK |                    |                   |Seq = 684244446 Ack = 2832617062
|           |(62086)  <------------------  (4377)   |                   |
|0.100008785|         ACK      |                    |                   |Seq = 2832617062 Ack = 684244447
|           |(62086)  ------------------>  (4377)   |                   |
|0.100008857|         SYN, ACK |                    |                   |Seq = 2664371816 Ack = 2950880130
|           |(14594)  <------------------------------------------  (4377)  |
|0.100008857|         ACK      |                    |                   |Seq = 2950880130 Ack = 2664371817
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100008923|         PSH, ACK - Len: 500           |                   |Seq = 2832617062 Ack = 684244447
|           |(62086)  ------------------>  (4377)   |                   |
|0.100009388|         PSH, ACK - Len: 500           |                   |Seq = 2950880130 Ack = 2664371817
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100017545|         ACK      |                    |                   |Seq = 684244447 Ack = 2832617062
|           |(62086)  <------------------  (4377)   |                   |
|0.100017617|         ACK      |                    |                   |Seq = 2664371817 Ack = 2950880130
|           |(14594)  <------------------------------------------  (4377)  |
|0.100018483|         ACK      |                    |                   |Seq = 684244447 Ack = 2832617562
|           |(62086)  <------------------  (4377)   |                   |
|0.100018948|         ACK      |                    |                   |Seq = 2664371817 Ack = 2950880630
|           |(14594)  <------------------------------------------  (4377)  |
|0.100020865|         ACK - Len: 1448               |                   |Seq = 684244447 Ack = 2832617562
|           |(62086)  <------------------  (4377)   |                   |
|0.100020865|         ACK      |                    |                   |Seq = 2832617562 Ack = 684245895
|           |(62086)  ------------------>  (4377)   |                   |
|0.100022089|         ACK - Len: 1448               |                   |Seq = 2664371817 Ack = 2950880630
|           |(14594)  <------------------------------------------  (4377)  |
|0.100022089|         ACK      |                    |                   |Seq = 2950880630 Ack = 2664373265
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100023313|         ACK - Len: 1448               |                   |Seq = 684245895 Ack = 2832617562
|           |(62086)  <------------------  (4377)   |                   |
|0.100023313|         ACK      |                    |                   |Seq = 2832617562 Ack = 684247343
|           |(62086)  ------------------>  (4377)   |                   |
|0.100024537|         ACK - Len: 1448               |                   |Seq = 2664373265 Ack = 2950880630
|           |(14594)  <------------------------------------------  (4377)  |
|0.100024537|         ACK      |                    |                   |Seq = 2950880630 Ack = 2664374713
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100025761|         ACK - Len: 1448               |                   |Seq = 684247343 Ack = 2832617562
|           |(62086)  <------------------  (4377)   |                   |
|0.100025761|         ACK      |                    |                   |Seq = 2832617562 Ack = 684248791
|           |(62086)  ------------------>  (4377)   |                   |
|0.100026985|         ACK - Len: 1448               |                   |Seq = 2664374713 Ack = 2950880630
|           |(14594)  <------------------------------------------  (4377)  |
|0.100026985|         ACK      |                    |                   |Seq = 2950880630 Ack = 2664376161
|           |(14594)  ------------------------------------------>  (4377)  |
|0.100027576|         PSH, ACK - Len: 656           |                   |Seq = 684248791 Ack = 2832617562
|           |(62086)  <------------------  (4377)   |                   |
|0.100027576|         ACK      |                    |                   |Seq = 2832617562 Ack = 684249447
|           |(62086)  ------------------>  (4377)   |                   |
|0.100028166|         PSH, ACK - Len: 656           |                   |Seq = 2664376161 Ack = 2950880630
|           |(14594)  <------------------------------------------  (4377)  |
|0.100028166|         ACK      |                    |                   |Seq = 2950880630 Ack = 2664376817
|           |(14594)  ------------------------------------------>  (4377)  |
```

Figure 3: QueryResponseApplication TCP flow diagram as observed at the query host.

No "think time" is currently simulated by responders in between receipt of a query and commencing transmission of the response. Implementation would be trivial, but determining an appropriate distribution less so and was beyond the scope of work required for this thesis.

4.3 A VOQ Ethernet Switch Model for CLUES

The CLUES VOQ Ethernet switch model is an almost complete rewrite of the standard ns-3 BridgeNetDevice model. The forwarding logic was heavily modified, while the Media Access Control (MAC) aspects of the model were not touched. The key technical design choices to model buffers as fixed size cells, and backplane latency as a fixed time interval added to frames after leaving the receive port's ingress VOQ, both stemmed from the prior work of Lincoln Dale[8].

Mtu	Maximum transmission unit supported by the switch ports
EnableLearning	MAC address learning
ExpirationTime	Length of time a learned MAC address remains cached
RxQueueCellsPerPort	Number of ingress VOQ cells
TxQueueCellsPerPort	Number of egress FIFO queue cells
BytesPerCell	Number of bytes per cell buffer
BackplaneLatency	Time delay added to frames on forwarding from ingress VOQ to egress queue
TxQueueMgmtScheme	Name of the queue management scheme used to determine when frames can be forwarded from ingress VOQ to egress queue

Table 3: ns-3 BridgeNetDevice model configuration options.

The model offers the configuration options detailed in Table 3.

BridgeNetDevice groups a set of NetDevice objects to create a multi-port bridge. The ingress receive hook of each port is set to the BridgeNetDevice `ReceiveFromDevice` method, which is the entry point for packets to the bridge control plane. Unhelpfully for the BridgeNetDevice use case, the NetDevice receive hook passes the decapsulated packet. For the CsmaNetDevice (Ethernet port) objects used by the CLUES model, this means the bridge control plane sees IP packets instead of Ethernet frames. Working around this wrinkle added some complexity to the implementation that ideally should be improved.

The Queue model was modified to understand cells as a unit of buffer, and a

[8]Lincoln used ns-3 and a modified BridgeNetDevice model to conduct some proprietary research for Arista Networks, Inc. and shared his code with me.

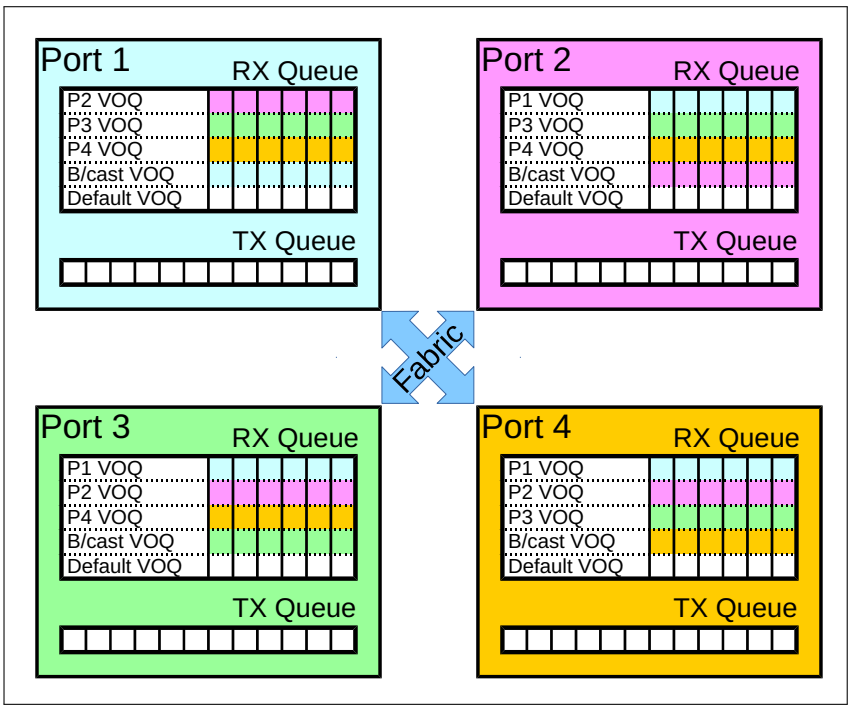


Figure 4: High-level logical view of a four port CLUES switch.

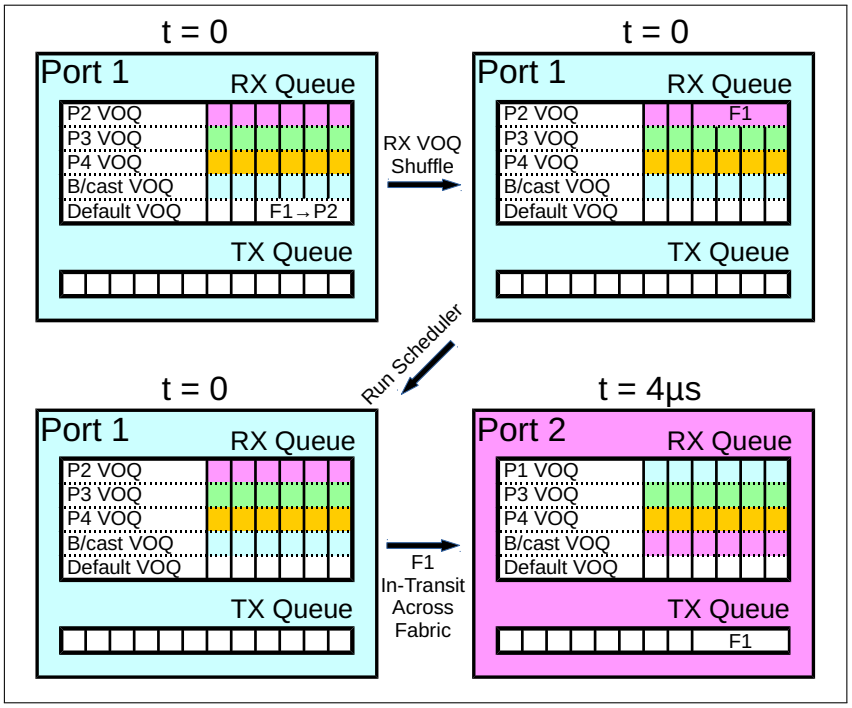


Figure 5: Key bridge machinations on receiving a 4-cell frame (F1) to be forwarded out port 2 (P2), with 4 µs port-to-port forwarding delay configured.

new MultiQueue model was created to underpin the VOQ functionality required for the ingress queue. A MultiQueue instance functions as a set of drop-tail sub-queues sharing a single resource limit configured in units of packets, bytes or cells. Sub-queues are selected using an arbitrary Object pointer, with `NULL` used as the default sub-queue. Figure 4 shows the logical view of a four port CLUES switch and Figure 5 shows the key steps involved in forwarding a frame.

The CsmaNetDevice model was altered to add a receive queue and enqueue to it on ingress. Host nodes enqueue and immediately dequeue so are not materially affected by the change. A switch node sets the receive queue for its CsmaNetDevice instances to the new MultiQueue model and enqueues received frames in the default sub-queue prior to decapsulation and calling the receive hook. If successfully enqueued in the default sub-queue, sufficient space exists to receive the frame and therefore receive hook processing will continue.

After determining that the received frame does need to be forwarded to one or more egress ports, `ReceiveFromDevice` perpetrates a gross layering violation to implement the VOQ functionality. The frame enqueued by CsmaNetDevice is dequeued from the default sub-queue and immediately enqueued again, but this time using the egress port's NetDevice pointer as the sub-queue address. The overall ingress queue occupancy remains the same after this shuffle, but the frame is now logically queued per its target egress port.

Efficient handling of broadcast frames posed a special challenge, which was addressed by designating the ingress port's NetDevice pointer as the broadcast sub-queue address. This works because an ingress queue should never legitimately need to queue frames for transmission out the same port they were received on.

With the frame correctly ingress queued, the bridge scheduler is invoked using the `BridgeScheduler` method. The scheduler implements the logic responsible for forwarding frames from ingress queues to egress ports per the Figure 6 flow chart. Every port is iterated as a candidate egress port, and for each, the appropriate VOQ sub-queue of every other port's ingress queue is serviced in a round-robin manner until empty or the egress port's forwarding limit is reached.

The scheduler maintains a persistent record of the ingress sub-queue to be serviced first for each egress port at the next invocation, so that correct round-robin servicing of VOQs is maintained. The design decision to restart scheduling from the VOQ that failed to be serviced last time removes the opportunity for scheduling starvation caused by other VOQs containing smaller frames. The trade off is that

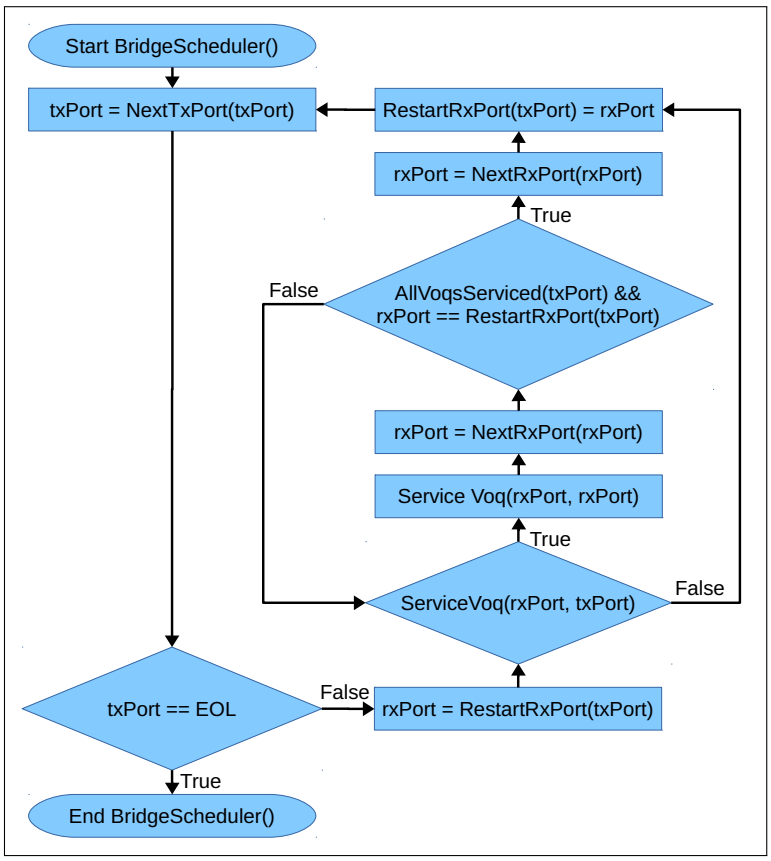


Figure 6: Bridge scheduler pseudo-code flow chart.

transmit opportunity across the "fabric" can be lost if a frame in another VOQ could have been forwarded during the current invocation. The broadcast sub-queue is serviced on more of a best-effort basis and does not influence the scheduling logic if a broadcast frame fails to be serviced.

ServiceVoq() considers a VOQ successfully serviced if the queue is empty, or the frame at the head of the queue was dequeued and forwarded across the fabric to the egress port. Fabric control logic is implemented in the CanBeginForwarding() method, which is a wrapper around the scheme selected using the "TxQueueMgmtScheme" configuration option. Two schemes are implemented, namely "cellcredit" and "oracle".

The "cellcredit" scheme approximates the way commodity switches operate, by treating the available egress queue cells as fabric admission control tokens. The token count is decremented as frames are admitted to the fabric and incremented as frames are dequeued at the egress port for transmission. Frames are admitted only if the available token count equals or exceeds the number of frame cells. When

selected, this scheme invokes the scheduler from the egress port's dequeue callback on commencing transmission of a frame i.e. after some tokens have been released. The scheme does not exploit a range of information available to it, such as the fabric forwarding latency. This can lead to lost transmission opportunity depending on the balance between fabric forwarding latency, egress port transmission rate, queue size, and queue occupancy.

The "oracle" scheme is an entirely different beast, (ab)using its access to various simulated objects' internals to perform a detailed queue projection. The scheme is able to instantiate an optimal pipeline across the fabric that eliminates the potential for transmission opportunity loss, unlike the "cellcredit" scheme. This scheme's ability to calculate precisely when a frame should leave the ingress VOQ allows it to create a future simulator event to invoke the bridge scheduler at the correct time for the frame to be forwarded.

4.4 A FreeBSD-based Virtualised Network Stack for CLUES

The CLUES stack offers several notable improvements and contributions to NSC and NSC-based simulations.

4.4.1 Port Layout

The port is structured to maximise ease of development, configuration flexibility and maintainability. Upstream source code lives in the `pristine` port directory as a virgin Subversion working copy, managed like any other FreeBSD source tree working copy. Updating the upstream source files or expanding the port to include additional source files is therefore a simple Subversion update operation without the need for any complex merging and conflict resolution.

A small number of pristine files require some manipulation for the stack to be successfully compiled, and this is achieved by preferencing the `override` port directory in the compiler's search path. Override files are therefore preferentially injected into pristine files by the preprocessor, providing a cleanly separated means with which to achieve the required manipulation.

FreeBSD kernel configuration influenced build files reside in the `kernconf` port directory, grouped in such a way as to allow future flexibility to build different "flavours" of stack using different kernel configurations.

The `glue` port directory contains the support code required to interface the stack with the NSC API, and emulate kernel infrastructure required by the upstream code in `pristine`. Many of the glue source files require customised build configuration on account of the subtleties inherent in realising this interface, and are special cased by the build script for this reason.

Finally, the `testharness` directory contains a build target that links a test harness executable against the stack library. The build script builds the test harness immediately after the library to identify any linkage problems.

4.4.2 Modifications to NSC and its integration with ns-3

The NSC API and integration with ns-3 were modified for CLUES to address a number of issues and opportunities identified during the porting work.

Reorganisation of callbacks

NSC provides a set of callback function pointers which are organised in a somewhat ad hoc manner, made worse by the callback changes detailed below. Callback functions were reorganised into a new set of structs per Listing 1 to clearly group callbacks that pertain to the library as a whole, an individual stack instance or an individual socket instance respectively. The appropriate struct type is passed at library initialisation, stack creation or socket creation time, with pointers set by the simulator.

The FRandom callback was removed rather than migrated into `NscLib2SimCallbacks` as it was unused and would likely require a different definition if it were ever to be reinstated.

High-precision event-timer interface

Kernels have traditionally provided a fixed interval ("tick") periodic wake up mechanism that kernel subsystems use to schedule events. Events from any source are coalesced into the appropriate queue of future events scheduled n ticks from now. Achieving sub-tick resolution is not possible, instead requiring that the tick rate be increased to match the required interval resolution. Higher tick rates increase CPU utilisation and system power consumption, potentially with minimal or no benefit if the higher rate is only required sometimes or by a subsystem that is inactive.

Contemporary kernels tend to take advantage of the improved hardware timers

```
#define CLOCK_MASK              0x0000000F
#define FRAC_SHIFT              4
#define FRAC_MASK               0x000000F0
#define FRAC_NS                 (1 << FRAC_SHIFT)
#define FRAC_US                 (2 << FRAC_SHIFT)
#define FRAC_BT                 (3 << FRAC_SHIFT)
#define FLAG_MASK               0xFFFFFF00
#define FLAG_SHIFT              8
#define FLAG_INTERPOLATE        (0x01 << FLAG_SHIFT)

typedef void (*FEventFired)(void *);

struct NscLib2SimCallbacks {
  virtual ~NscLib2SimCallbacks() {}
  virtual void evsched_cb(uint64_t when, uint32_t units,
     FEventFired func, void *ctx) = 0;
  virtual void gettime_cb(uint32_t flags, int64_t *sec, int64_t
     *frac) = 0;
};

struct NscStack2SimCallbacks {
  virtual ~NscStack2SimCallbacks() {}
  virtual void iftx_cb(const void *data, int len) = 0;
};

struct NscSocket2SimCallbacks {
  virtual ~NscSocket2SimCallbacks() {}
  virtual void sosnd_cb() = 0;
  virtual void sorcv_cb() = 0;
};
```

Listing 1: Revised NSC callbacks.

available in modern systems that allow them to run in a high-precision event-driven rather than periodic wake up mode of operation. So called "tickless" mode kernels allow kernel subsystems to specify with high-precision when they wish to wake up, as well as how precisely the requested time needs to be honoured. This removes the reliance on a system-wide periodic timer for event management, maintains the ability to coalesce events, and better caters for high-precision requirements only when needed.

NSC uses the traditional tick-based approach driven from the simulator's event loop. It therefore suffers from the associated interval resolution and computational overhead issues as a result. NSC was adapted for CLUES by replacing the former mechanism with the `evsched_cb` high-precision event-timer interface detailed in Listing 1. The interface allows stacks to insert arbitrary callback function events

into the simulator event loop some number of (nano|micro|milli)seconds into the future, thereby allowing them to internally manage their own time keeping needs.

Socket upcalls

The NSC `wakeup` callback functions as a library-wide indiscriminate mechanism for a stack to inform the simulator that a socket generated an upcall. By not providing the simulator with the identity of the socket, the simulator was forced to check every NSC socket associated with the stack for actionable changes.

NSC was adapted for CLUES by replacing the coarse `wakeup` callback with the new `NscSocket2SimCallbacks` send and receive upcall callbacks detailed in Listing 1. Socket upcalls generated within the stack for a given socket now propagate all the way up to the simulator's corresponding NSC wrapper socket.

Improved time keeping integration

NSC provides the `gettime` callback function for stacks to query the simulator's virtual clock, returning the elapsed virtual run-time as seconds and microseconds. The interface does not provide sufficient flexibility for a range of use cases including the low latency simulation scenarios of interest for CLUES.

The interface and ns-3 implementation were completely redone for CLUES. The interface was revised as `gettime_cb` together with the flags shown in Listing 1. The new interface uses the same `CLOCK_UPTIME` and `CLOCK_REALTIME` defines from the conceptually related POSIX `clock_gettime` API.

The new ns-3 implementation provides three key features. It can return fractional seconds in a range of resolutions per the `FRAC_` flags. It can return time adjusted relative to the underlying system's wall clock if `CLOCK_REALTIME` is specified. Finally, it can use the CPU's Time Stamp Counter (TSC) to interpolate between step increments of the virtual clock if `FLAG_INTERPOLATE` is specified.

The interpolation feature provides an option to fuzz the clock by amounts proportional to the code execution time between clock reads, which can potentially improve experimental realism. The improvement stems from introducing a source of noise into clock reads, and avoiding quantisation effects associated with multiple clock reads made between virtual clock increments all returning identical time.

The implementation guarantees that time will never go backwards by internally latching the largest interpolated clock read returned, but would benefit from addi-

tional refinement. It does not currently look forward in the event queue to determine the interpolation upper bound. This makes it possible that the next step of the virtual clock leaves the true virtual clock behind the previously returned interpolated clock read. In this situation the implementation returns the latched interpolated clock read to ensure time does not go backwards, but in so doing returns a clock read ahead of where it should be at that time during the simulation.

Route insertion

On account of the NSC API's choice to exchange fully formed IP packets between stack and simulator, the stack's routing table needs to have a route to every destination in order for it to emit packets. This is of course a trick played on the stack, given that the simulator strips the IP header from the packet and prepends its own based on its internal routing logic. To that end, NSC provides the `add_default_gateway` function so that each stack instance's routing table can be primed with the minimal piece of configuration required to trick the stack. However, this mechanism is insufficient for a range of statically routed topologies that require explicit routes in addition to, or instead of, a default gateway.

```
virtual void add_default_gateway(const char *addr) = 0;
virtual void route(const char *dst, const char *mask, const char
    *gw) = 0;
```

Listing 2: NSC routing table interface.

NSC was augmented with a simple `route` interface for CLUES per Listing 2. The interface is functional but too restrictive, as it currently only accommodates additions to the routing table. However, it is underpinned by code that is capable of full routing table manipulation including metrics, and the interface should be extended to allow the full range of manipulations to be expressed. `add_default_gateway` was retained for posterity even though `route` provides a superset of functionality.

ns-3 was augmented to use the new `route` interface in place of `add_default_gateway`. The list of shortest prefix routes are computed for the entire simulation topology by enumerating the ns-3 `Ipv4GlobalRouting` model instance, and the list is then injected into the stack's routing table using the `route` interface.

4.4.3 Port Internal Key Points of Interest

The port takes in a significant amount of FreeBSD kernel infrastructure in an effort to maximise realism by minimising API emulation and modifications required to pristine sources using the overrides mechanism. Synchronisation and scheduling primitives, kthreads, sysctl, and the Universal Memory Allocator (UMA) are some of the key pieces of infrastructure included, functioning almost entirely in their upstream form.

64-bit

The stack is built with the 64-bit x86 Instruction Set Architecture (ISA) (FreeBSD's "amd64" architecture) and that is currently the only option. Care and attention were given to the architecture of the build script to make supporting builds of different architectures from the same code base possible with minor effort.

Library initialisation

The simulator calls `nsc_init_lib` after dynamically loading the stack library, and this is used to trigger a mock kernel boot by calling `freebsd_so_init`.

Kernel modules

Code implemented as a FreeBSD kernel module works seamlessly when compiled into the CLUES stack, functioning as if it had been loaded by the boot loader prior to kernel start up. Support for dynamic modules implemented as separate shared libraries does not currently exist, but could be added with minor effort.

Memory allocation

FreeBSD's UMA slab allocator is used for most kernel code allocation requirements, and is included in the port to provide complete similarity with real kernel allocations. A mix of `mmap/munmap` and `posix_memalign/free` service the needs of UMA and a small number of other kernel subsystems that utilise non-UMA sourced allocations. Care has been taken to return allocations with the same alignment as real kernel allocations.

Stubs and globals

Functions which contain unrequired code and functionality but which are required by name for linking the stack reside in `glue/stubs.c`. Stubs not called from any actively used code paths explicitly terminate the application if invoked to aid with debugging. Stubs known to be called from actively used code paths but which do not need to perform their regular functions implemented in their upstream equivalents are left empty, or return an appropriate value if required.

Global variables referenced by pristine code but defined in source files not included in the port reside in `glue/globals.c` and are initialised to appropriate values where required.

VNET

The stack's integration of VNET with NSC allows an arbitrary number of stacks to be dynamically instantiated at run-time, negating the need for NSC's "globaliser" tool [248]. Initialising a new stack instance allocates a 48b `struct vnet` using FreeBSD's `vnet_alloc` and stores a pointer in the stack's `vnetstack` member variable. The VNET allocation embeds in it a copy of the "vnet" linker set of virtualised variables, which amounts to 16344b of memory per VNET and brings the total memory footprint per VNET to 16392b for CLUES v1.0.

Integrating and utilising VNET in this way completely does away with the build time, shared library size and fixed maximum number of stack instance limitations associated with the "globaliser" [248]. Further memory and CPU utilisation scalability improvements stem from the majority amount of infrastructure that is shared between all VNETs. The indirection cost associated with VNET virtualised variables is on the same order as "globalised" variables, therefore making the use of VNET an overall flexibility and scalability improvement.

Kernel thread emulation

The FreeBSD kernel utilises a number of permanent kthreads, scheduled periodically or via an event trigger, to perform various important functions. Retaining the same division of labour in the port was very desirable in order to maintain similarity of operation with real kernels. Porting the "kthread" infrastructure to work in the single threaded ns-3 context without preemption required some creative maneuvering. I devised a run-to-completion model using regular pthreads to back each kthread,

along with a custom "kernel scheduler" residing in `glue/sched_nsc.c` to manage the scheme.

The unmodified pristine code creates kthreads and schedules them using `sched_add` per normal. The scheduler maintains a list of one-to-one mappings between each kthread and its corresponding persistent pthread using a `struct pth2td`. If `sched_add` finds an existing mapping for the kthread it simply returns. For a previously unscheduled kthread, it creates a new pthread and mapping on demand, and starts the pthread in the scheduler's `pthread_entry` function. Each new mapping also contains a condition variable `cv` and mutex `cv_mtx` which are initialised and the mutex locked in `pthread_entry` prior to executing the thread's target function.

The target function runs and eventually calls, either directly or indirectly via a low level kernel API, `sched_switch` to yield the CPU to the next runnable thread. `sched_switch` signals the mapping's `cv` using `pthread_cond_signal` (for reasons to be explained shortly) and then quiesces the thread using `pthread_cond_wait`, which unlocks `cv_mtx` and waits for a call to `pthread_cond_signal` from another context to awaken it.

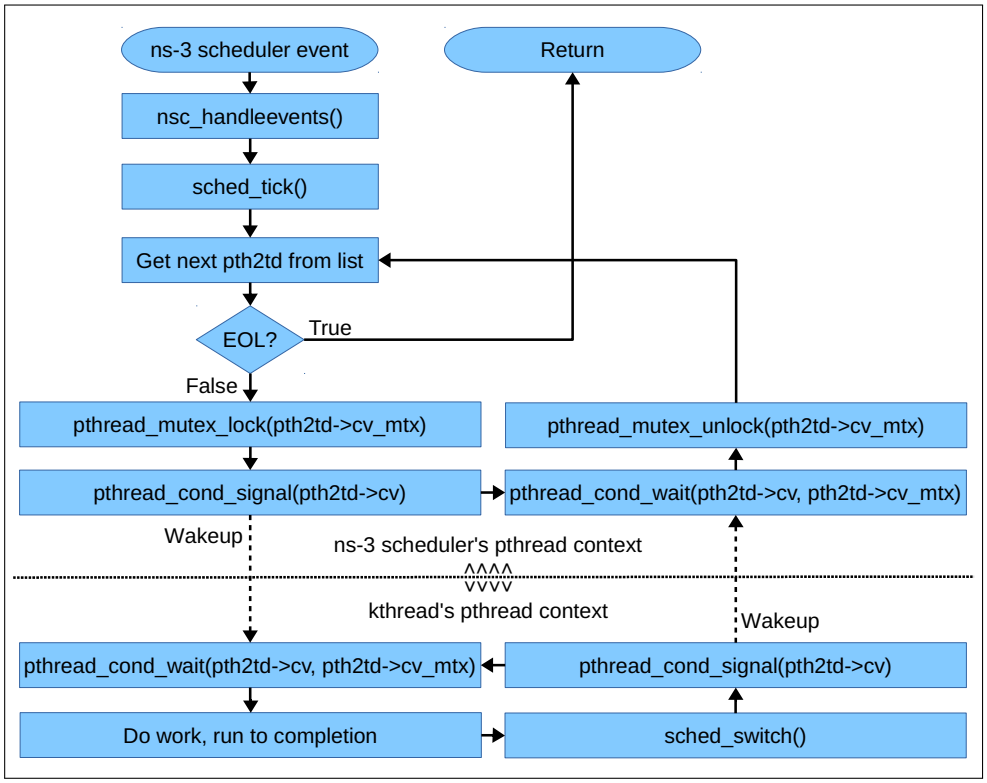


Figure 7: kthread emulation.

Awakening quiesced threads is driven by the simulator executing callback events that were previously scheduled by the stack using the `evsched_cb` NSC interface. Figure 7 shows the key interactions.

The port indiscriminately uses `nsc_handleevents` as the callback function for all stack scheduled events. It is modelled on FreeBSD's `handleevents`, which drives kernel machinations on receipt of hardware event timer interrupts. As one of its actions, `nsc_handleevents` calls `sched_tick`, which implements the final key aspect of the kthread emulation scheme.

Executing in the context of the main simulator thread, `sched_tick` walks the list of mappings and for each, acquires `cv_mtx`, signals `cv` using `pthread_cond_signal` and then waits on `cv` using `pthread_cond_wait`. Each pthread awaiting reanimation in `sched_switch` awakens on receiving the signal and runs to completion, again returning to `sched_switch`. The `cv` signal from `sched_switch` triggers `sched_tick` to progress through each mapping in the list. Note that at no point do multiple threads execute concurrently in the stack as a result of this scheme's implementation. On return from `sched_tick`, all kthreads will have run to completion once.

The override files mechanism detailed earlier is used to redefine the `curthread` macro in `pcpu.h` to return the result of the `sched_getcurthread` function. `sched_getcurthread` uses `pthread_self` to find the `struct pth2td` associated with the currently executing pthread. The FreeBSD kthread's `struct thread` associated with the found `struct pth2td` is then returned, thereby allowing the pristine kernel code to remain oblivious to the implementation details.

The lack of concurrency is the single biggest issue affecting realism with the port. An entire class of dynamic behaviours and bugs observable when running with concurrency in a real kernel vanish in the CLUES stack. In theory a multi-threaded ns-3 scheduler could be integrated into CLUES to address this issue and some relevant prior work does exist [260]. However, it would be a significant undertaking within ns-3 given the lack of attention to concurrency in the existing models and infrastructure.

Another issue is that without preemption, the run to completion implementation allows kthreads to complete an unbounded amount of work compared with normal scheduling quanta in a real kernel. It also runs threads with potentially very different scheduling intervals to those in a real kernel because NSC kthreads depend on the frequency with which `sched_tick` is called. A number of possibilities exist as future work to improve the realism of these aspects of the kthread emulation scheme.

4.5 Piecing Everything Together

The CLUES v1.0 "incast" simulation serves as a practical instantiation of how to use the CLUES components to investigate a data centre specific issue. Per Figure 8, a cluster of host nodes consisting of a single designated querier and one or more responders is connected in a star topology using full-duplex switched Ethernet via a central VOQ switch node. Host nodes run the NSC FreeBSD-based TCP stack and query-response application to generate a barrier-synchronised TCP workload, each in accordance with its role.

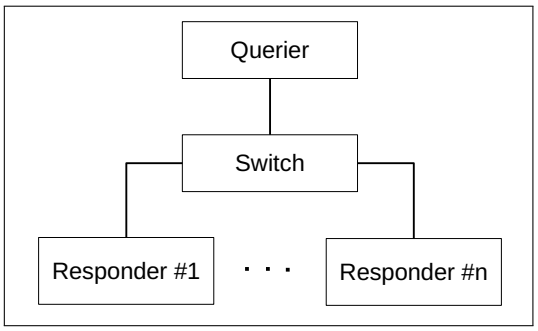


Figure 8: Topology created by the "incast" simulation.

The key ns-3 models and high-level simplified groupings used in the simulation are shown in Figure 9. Nodes predominantly function as logical containers for other models. The CsmaChannel model forms the communications link between node CsmaNetDevice models, simulating the experiment's configured propagation delay and transmission rate. The NSC library integration is primarily contained in the NscTcpSocketImpl and NscTcpL4Protocol models, which interact with the library (and vice versa) using the NSC API and callback function pointers. ns-3 applications function similarly to regular network applications by using a Berkeley-sockets-like API to send and receive data.

Leveraging the CLUES "incast" simulation as a suitable experimental base, let us now turn our attention to investigating the characteristics and dynamics associated with the TCP incast congestion phenomenon.

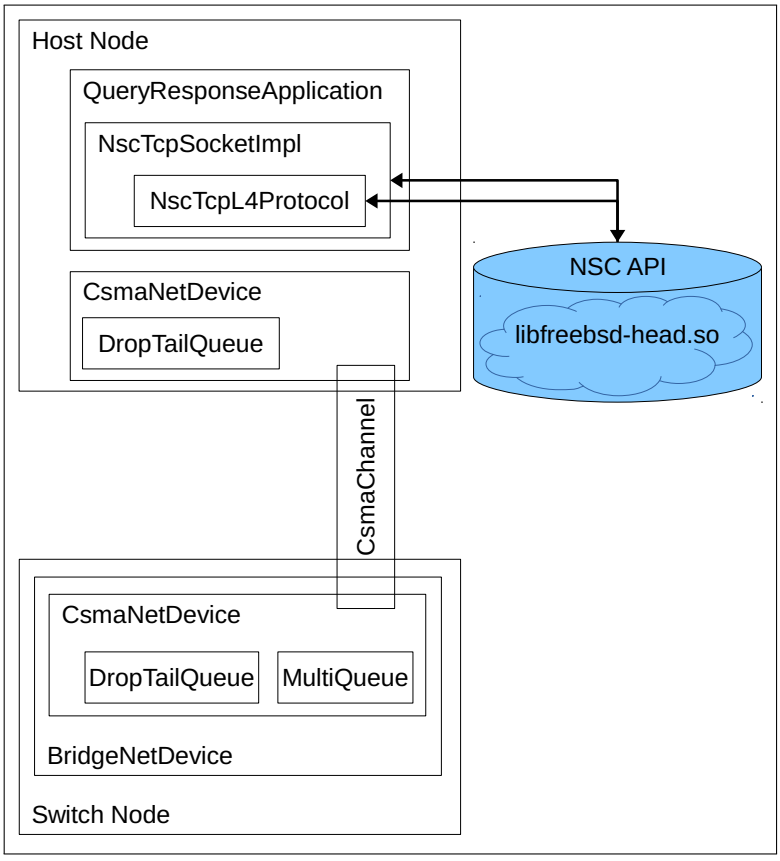


Figure 9: Key models and simplified groupings used in the "incast" simulation.

5 An Investigation of TCP Incast Congestion

The investigation begins with a discussion of the key operational aspects of TCP as they relate to incast congestion. The general discussion is followed by enumeration of the experimental methodology employed, and presentation of experimental results together with associated analytical discussion.

5.1 General Discussion

A key difference between incast and regular congestion in statistically multiplexed networks is that the application layer workload and communications strategy inherently causes the correlation that leads to oversubscription. This fact is decoupled from the underlying transport, which introduces its own complexities when dealing with the effects of congestion like queue delay and packet loss.

The TCP throughput collapse pathology is well understood to be caused by the loss of transmission opportunity associated with RTO events. As discussed in 2.4.2, TCP intentionally calculates a conservative RTO interval per RFC 6298 and imposes a minimum value, RTO_{min}. If a situation occurs that requires an RTO to recover from, a sender will remain stalled until the retransmit timer fires.

With multiple concurrent senders in an incast communication scenario, other senders could continue to make progress and keep the pipeline full during the RTO stall. Depending on their send windows, they may or may not be able to fully utilise the contended link during the stall, but this does not in and of itself lead to aggregate throughput collapse. Rather, collapse occurs when either multiple senders stall concurrently, or when a stall period far exceeds the intrinsic completion time had no stall occurred. The former is often true due to the nature of incast congestion interacting with switch queues, and the latter is also typically true due to TCP's RTO_{min} and 1 ms granularity RTT measurement resolution.

The likelihood of multiple senders stalling in parallel is high because switches typically service queues of the same priority in round-robin fashion, which distributes drops across senders. As the senders' responses are correlated in time, a scenario leading to a RTO for only one sender at a time is unlikely. This aggregate loss of transmission opportunity is what can bring about aggregate throughput collapse.

The other mutually inclusive issue is the disparity between a path's true and measured RTT affecting the RTO interval calculation. RFC 6298 stipulates that the RTO interval is calculated as $RTO = max(RTO_{min},\ SRTT + 4 \times RTTVAR)$,

where SRTT and RTTVAR are weighted moving averages of the path RTT and RTT variance respectively. RTO_{min} is specified as 1s, but many stacks make it tunable and default it to a value on the order of tens or hundreds of milliseconds. As discussed in 2.4.3, TCP implementations do not typically measure RTT with sub-millisecond resolution. Therefore even if the RTO_{min} is lowered to 1 ms or below, there is insufficient measurement resolution to compute a RTO interval below 1 ms.

For typical cluster and data centre networks with microsecond-scale path latencies, the retransmit interval remains permanently stuck to the 1 ms floor instead of adapting to the path. If a RTO is incurred with a 1 ms interval, the amount of transmission opportunity loss increases together with transmission speed and/or decreasing path latency. In isolation or in parallel across multiple senders, this also results in underutilisation and throughput collapse.

A related side note – even though incast is predominantly discussed in the context of high speed, low latency data centre networking, it does not depend on these network characteristics in any existential sense. In fact, as long as responses are correlated at timescales comparable to a network's latency, the conditions exist for incast to occur.

5.2 CLUES-based Experimental Methodology

The "incast" simulation detailed in 4.5 was used to conduct the experiments for this investigation of incast congestion. A single experiment consists of two phases shown in Figure 10: initialisation followed by experiment. The initialisation phase is further divided into sub-phases corresponding to start up initialisation, which takes place at time t=0, and pre-experiment initialisation, which takes place between time t=0 and the beginning of the experiment phase at time t=100 ms.

The start up initialisation sub-phase is comprised of the listed steps. Link configuration involves setting data rate and propagation delay. NIC configuration involves setting queue type to drop tail and size to the maximum value so that queue overflows only occur at the switch. VOQ switch configuration involves enabling fine-grained per-port logging, and setting the MTU, queue buffer cell size, receive and transmit queue size, port-to-port forwarding latency, and transmit queue management scheme. Configuring NSC involves loading the desired library and setting it as the stack on all host nodes. Configuring ping applications involves setting start and stop time, and ping interval. Configuring query-response applications involves setting start and stop time, number of queries, peer addresses, query and response

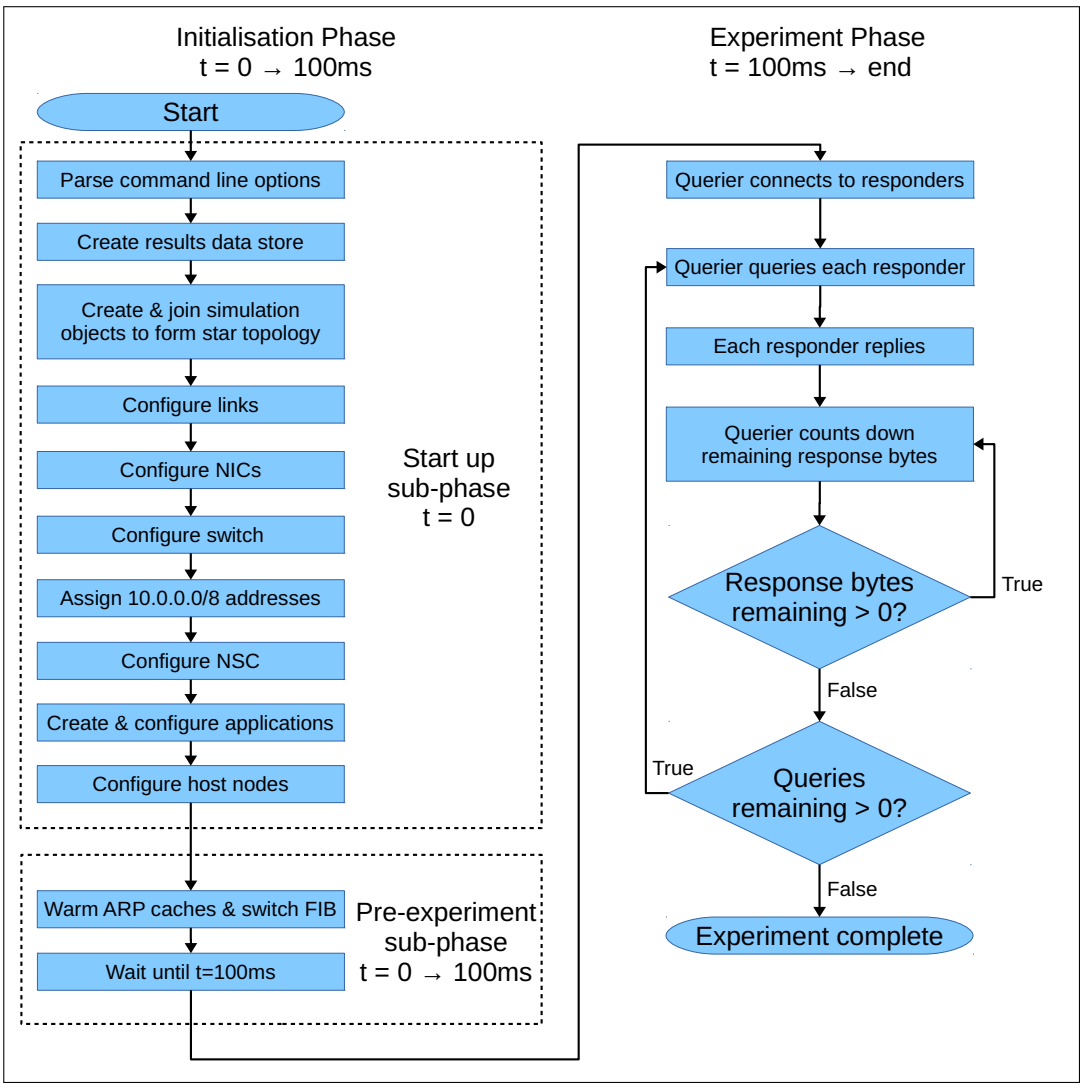


Figure 10: Flow chart showing experiment phases.

sizes. Configuring host nodes involves enabling per-packet pcap and Statistical Information For TCP Research (SIFTR) logging, and setting NSC stack sysctls.

The pre-experiment initialisation sub-phase consists of warming the switch's FIB and the Address Resolution Protocol (ARP) caches of all host nodes by way of each responder pinging the querier node. A 5 ms interval is used with an 11 µs staggered offset between responders (to avoid ARP incast!) until time t=20 ms i.e. responders typically manage three echo request/reply exchanges during the 20 ms window. The network is then allowed to fully quiesce until time t=100 ms, ready for the experiment phase to begin.

Barring any explicitly noted exceptions, experiments were specified in terms of

Variable	Run-time configurable?	Value Used
# query-response transactions	Y (`--queries`)	1
# query bytes	Y (`--query-size`)	500 B
# responders	Y (`--responders`)	mixed
# per-responder response bytes	Y (`--response-size`)	mixed
# switch RX VOQ cells	Y (`--rxqueue-cells`)	mixed
# switch TX cells	Y (`--txqueue-cells`)	24
# switch bytes per cell	Y (`--bytespercell`)	256 B
Switch forwarding latency	Y (`--backplane-latency`)	4 µs
Switch TX queue management scheme	Y (`--txqmgmt`)	"oracle"
Ethernet speed	Y (`--link-speed`)	mixed
Link propagation delay	Y (`--link-propdelay`)	134 ns
NSC library	Y (`--nsc-stack`)	"freebsd-head"
`net.inet.tcp.nagledelay` sysctl	Y (`--nagledelay`)	0 (disabled)
`net.inet.tcp.msl` sysctl	Y (`--msl`)	30 s
`net.inet.tcp.sendspace` sysctl	N	1048576
`net.inet.tcp.recvspace` sysctl	N	1048576
`net.inet.tcp.delayed_ack` sysctl	N	0 (disabled)
Node MTU	N	1500 B
Host node NIC (CsmaNetDevice) queue size	N	UINT_MAX

Table 4: CLUES incast simulation experiment variables.

the variables documented in Table 4. The link propagation delay models Cat 6A UTP cable which has a worst case propagation delay of 536 ns over 100 m [261], or 134 ns over 25 m as modelled for the experiments. The NSC FreeBSD stack configuration used default values except for the noted changes. TCP delayed acknowledgments were disabled to maximise timeliness of feedback and avoid the need to implement a mitigation technique for dealing with issues surrounding the delayed acknowledgement timeout.

The NSC FreeBSD stack was patched as described below to facilitate research into arbitrary timescale TCP behaviour, and the direct comparison between Chapter 5 and 6 experiments.

Stack variables related to RTT and RTT derivatives (e.g., RTO interval) had their types changed to the `sbintime_t` data type. The underlying data type of a

`sbintime_t` is `int64_t` i.e. a signed 64 bit integer, with the top 32 bits used to store seconds and bottom 32 bits for fractions of a second. The minimum resolution representable with this data type is therefore $\frac{1}{2^{32}}$s or approximately 233 ps. Code associated with these variables was revised to function in terms of the new data type, including a change to high resolution timers.

The stack's RTO_{min} (including the historical `net.inet.tcp.rexmit_slop` factor) was removed, thereby allowing the RTO interval to fully adapt to the available RTT measurement resolution. This made it possible to fully observe the effects of RFC 6298 [262] RTO interval calculation for all experiment scenarios.

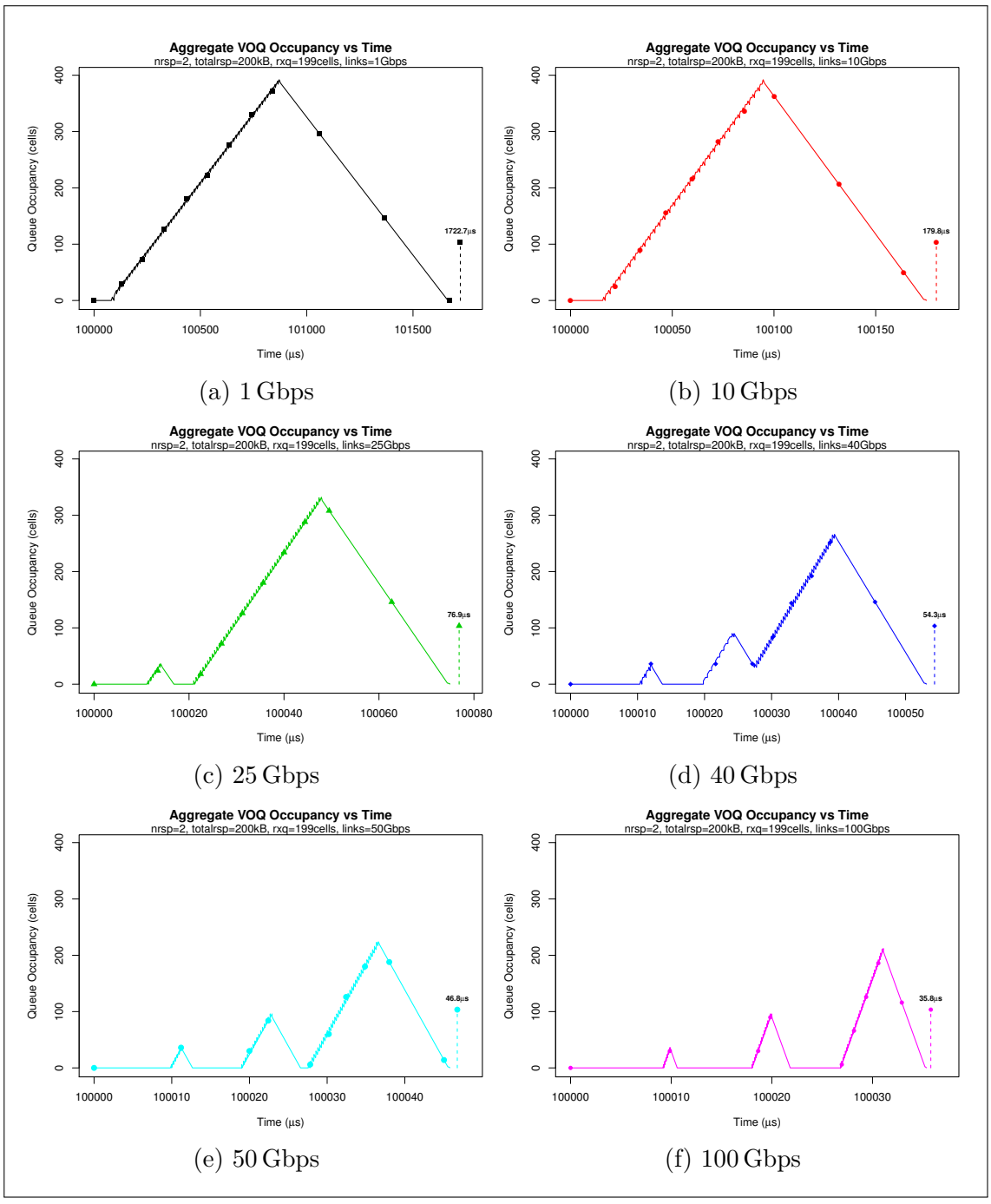
A sysctl to control the use of Nagle's algorithm was added to avoid having to implement the plumbing required to allow ns-3 applications to set the `TCP_NODELAY` socket option.

Finally, the scheme proposed in Chapter 6 was present, but bypassed for Chapter 5 experiments by way of a sysctl master control switch.

5.3 Exploring Cause and Effect

Consider a query-response transaction between a single querier and responder. With a non-blocking switch design and the querier connected at the same speed as the responder (both common practice), there will be no contention for transmit bandwidth towards the querier. The response ingress rate to the responder's switch port equals the egress rate from the querier's switch port, and therefore no standing queue would be built at the responder's switch port VOQ.

Now consider a similar scenario with the only difference being that the querier enlists two responders to answer the query, thus dividing the total amount of response data equally between both responders. Even though the total amount of data returned to the querier is the same, the responders return their portion in parallel. From the switch's perspective, the data arrives twice as fast as the single responder scenario and the ingress rate is double that of egress. The contention for transmit bandwidth towards the querier will therefore result in queue build up at each responder's switch port VOQ. If the imbalance between ingress and egress rates is sustained long enough relative to the upper VOQ limit, the switch will experience an incast congestion event resulting in packet loss.



(a) 1 Gbps (b) 10 Gbps

(c) 25 Gbps (d) 40 Gbps

(e) 50 Gbps (f) 100 Gbps

Figure 11: Aggregate VOQ occupancy versus time over 1, 10, 25, 50 and 100 Gbps networks.

5.3.1 Queue Occupancy Versus Ethernet Transmission Speed

Figure 11 plots the aggregate occupancy of all responder switch port VOQs over the course of a query-response transaction. The "lollipops" mark each experiment's application layer transaction completion time for clarity. Six different experiments show two responders (nrsp) returning a combined total of 200 kB of response data (totalrsp) over networks running at available data centre Ethernet speeds of 1, 10, 25, 40, 50 and 100 Gbps. All switch ports had 199 cells of VOQ buffering, which was the minimum required to avoid any packet loss.

Each experiment demonstrates some amount of queue build up, confirming the intuitive expectation that the incast phenomenon is not directly tied to network transmission speed; as long as responses are correlated at timescales comparable to a network's latency, the conditions exist for incast to occur. An important note with VOQ-style switch operation is that adding responders increases the effective queue size given that each ingress port independently buffers frames. With 199 cells of VOQ buffer per port, the aggregate number of frames which can be queued for a specific egress port is 398 cells worth, which is the high water mark visible for both the 1 Gbps and 10 Gbps experiments.

As network speed increases, the corresponding decrease in serialisation delay decreases the intrinsic path latency. This translates at the application layer to reduced query-response transaction completion times. The decreasing serialisation delay also reduces the queue high water mark on account of the gaps in transmission imposed by TCP's windowed flow control. At 50 Gbps and 100 Gbps, the queue was able to completely drain between each window of data.

The fixed number of responders for all experiments maintained a constant ratio between the impulse and impulse response. This manifests as a consistent rate of change in queue growth for all of the scenarios.

The remainder of this chapter will primarily focus on 10 Gbps and to a lesser extent 100 Gbps networks, given the former's prevalence in current commodity data centres and latter's emerging widespread availability.

5.3.2 Queue Occupancy Versus Number of Responders

Figure 12 also plots the aggregate VOQ occupancy over the course of a transaction, but with a varied number of responders. Five different experiments are shown in which the number of responders equally split 200 kB of response data between them

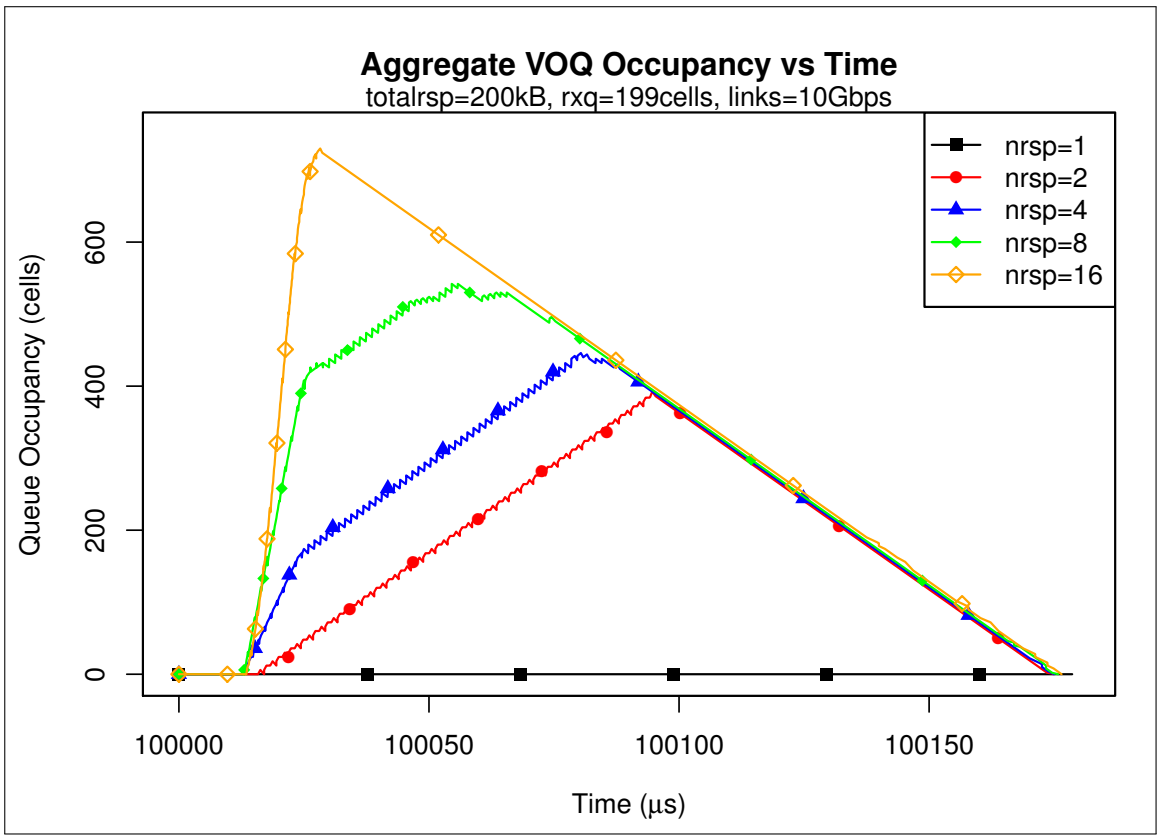


Figure 12: Aggregate VOQ occupancy versus time for 1, 2, 4, 8 and 16 responders.

i.e. each responder returned $\frac{200\,\text{kB}}{nrsp}$ of response data. All nodes were connected to the switch at 10 Gbps and all switch ports had 199 cells of VOQ buffering which was the minimum required to avoid any packet loss. The $nrsp = 2$ data series corresponds with Figure 11b.

As expected, increasing the number of responders increases the imbalance between ingress and egress rates, resulting in larger individual and aggregate VOQ utilisation. The minor reductions in transaction completion time as the number of responders increases is directly attributable to the additional parallelism. A small amount of extra transmission opportunity is gained from the switch scheduler by having more frames available earlier on during the transaction to fill the fabric forwarding pipeline.

The change of gradient visible during the queue growth period of the $nrsp = \{4|8\}$ experiments marks the point at which the initial window burst is complete and the responders transition to ACK-clocked slow start. For $nrsp = 16$, each responder is returning 12.5 kB of data which entirely fits within the initial window burst size of

14.5 kB. Consequently, this experiment exhibits a single gradient during the queue growth period. Conversely, the initial window burst for $nrsp = 2$ is small enough that after accounting for the switch forwarding pipeline, the initial window burst negligibly contributes to queue growth. The queue growth period for this experiment is instead dominated by ACK-clocked slow start, and therefore also exhibits a single queue growth gradient.

5.3.3 Comparing the Impact of Loss Recovery Mechanisms

The deliberate choice of per-port VOQ buffer size for the experiments shown in Figure 12 matches the ingress queue high water mark for the chosen set of parameters. The ingress queues were therefore a single cell away from an incast induced packet loss. If the threshold is crossed and packet loss occurs, TCP's ability to repair the damage and limit the associated negative performance impact depends heavily on a few important factors explored in Figure 13.

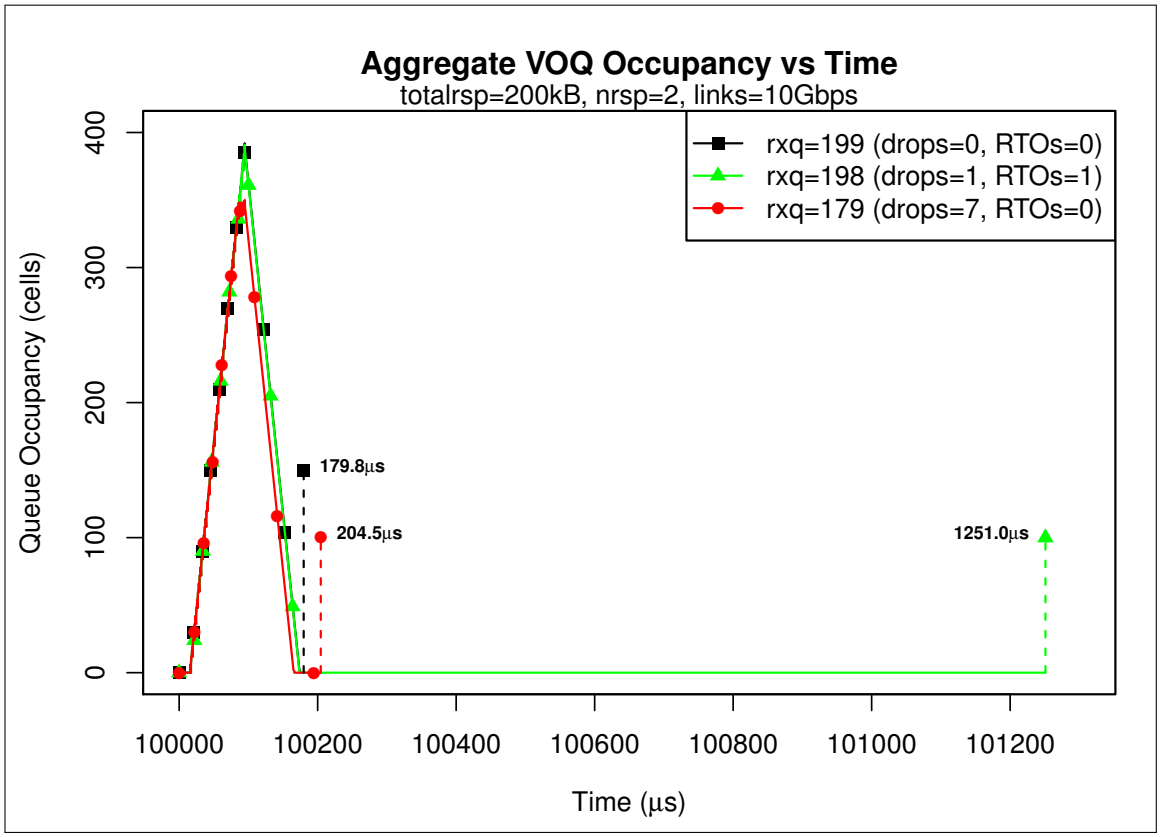


Figure 13: Aggregate VOQ occupancy versus time for 199, 198 and 179 cells per port VOQ buffering.

The Figure 13 $rxq = 199$ data series corresponds with the $nrsp = 2$ data series in Figure 12. By reducing the VOQ buffer size from 199 to 198 cells, a loss was induced at the tail end of one of the responders' data streams[9]. Figure 14 plots the evolution of the TCP sequence space during the experiment to help visualise what transpired.

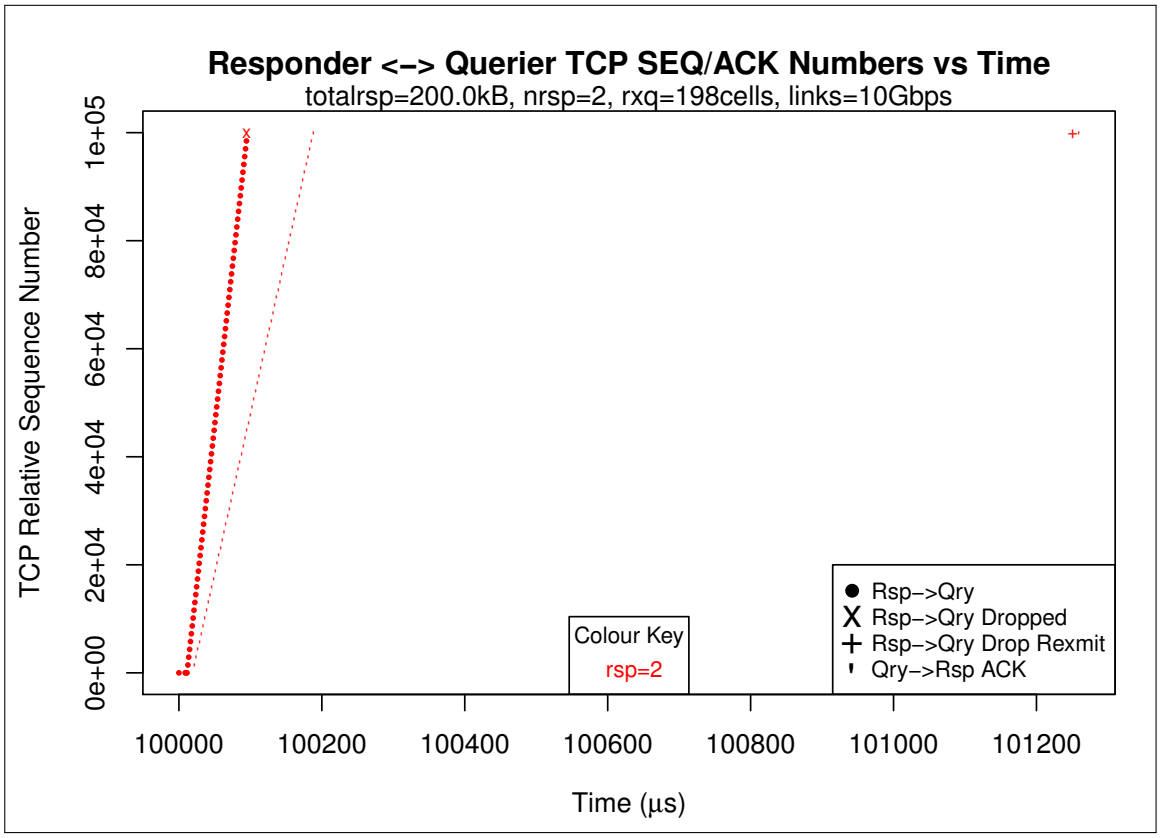


Figure 14: TCP sequence/acknowledgement numbers versus time, observed at the ingress switch port for the responder that experienced the RTO event.

Insufficient response data remained to be sent at the time of the loss, and the responder therefore never received the three duplicate ACKs required to trigger a fast retransmit of the lost packet. SACK was also of no use because no further packets arrived at the querier after the loss occurred, so no hole was detected and the querier only responded with cumulative partial ACKs up to the lost sequence number. The responder remained deadlocked at this point and had to wait for a RTO to trigger the retransmit that allowed the transaction to complete.

[9]Given that fine grained control of switch buffers is uncommon, it would have been more realistic to induce loss by increasing the response length. However, keeping the baseline completion time constant between experiments facilitates direct comparison and therefore simplifies discussion.

Returning now to Figure 13, it may seem counter intuitive that the $rxq = 179$ experiment with a total of 7 lost packets completed its transaction in an appreciably shorter period of time than the $rxq = 198$ single loss experiment. However Figure 15 provides useful insight to assist with the explanation.

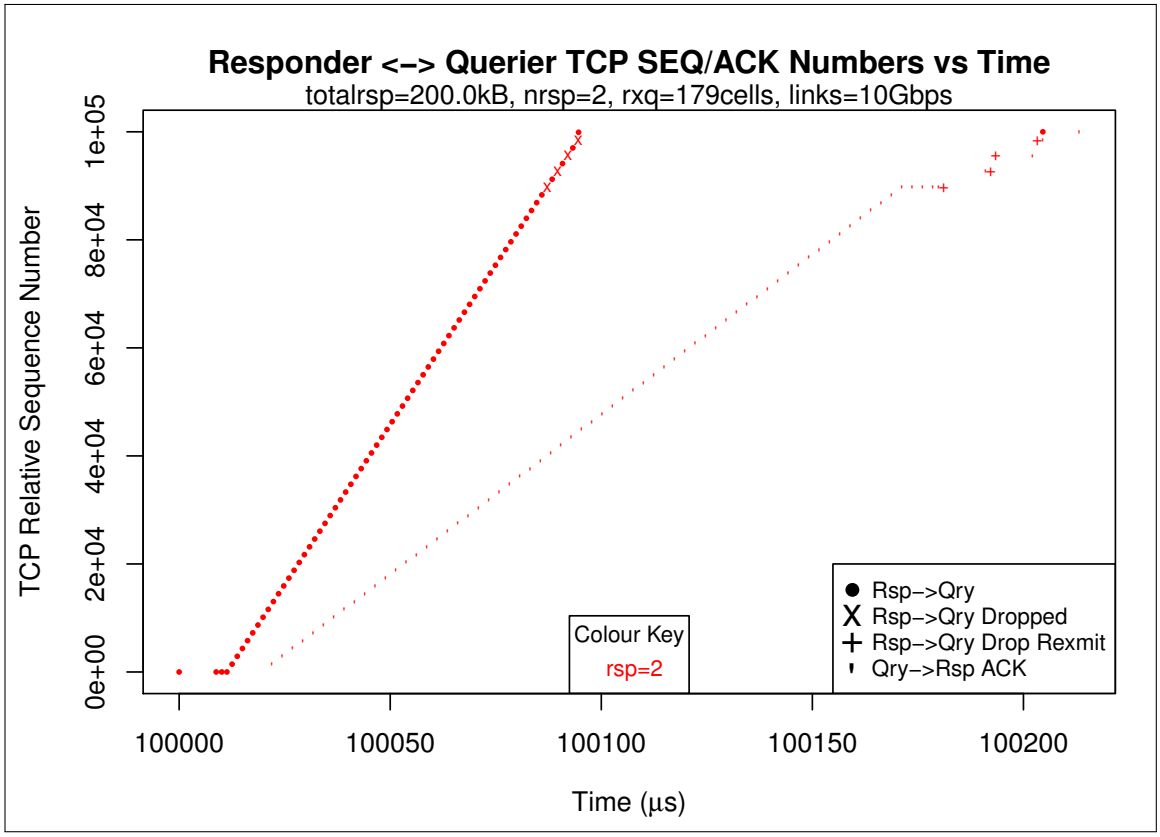


Figure 15: TCP sequence/acknowledgement numbers versus time, observed at the ingress switch port for the second responder.

At the point of VOQ saturation, the round robin switch scheduler is effectively causing every other packet sent by the responders to be dropped as it alternates between servicing each VOQ. This is clearly visible as the pattern of 4 dropped packets interspersed with 4 forwarded packets approximately 80 µs into the transaction. Unlike the single tail drop in Figure 14, the 4 forwarded packets after the first loss each triggered a duplicate acknowledgement, which are visible as the zero gradient portion of the ACK data series approximately 170 µs into the transaction. The series of 4 duplicate ACKs met the threshold of 3 required to trigger a fast retransmit and enter fast recovery.

Additionally, each of the 4 successfully forwarded post-loss packets allowed the querier to infer the losses from the gaps in the sequence space of the arriving packets.

The querier was therefore able to include SACK option data with each elicited duplicate ACK, and this allowed the responder to perform multiple retransmits per RTT during fast recovery. Without the SACK option data, the connection would have had to rely on standard NewReno loss recovery which only allows a single retransmit per RTT. Although the recovery process was quite involved and took 3 RTTs to complete, it was still an order of magnitude faster than relying on a RTO.

5.3.4 Measured Versus Path RTT

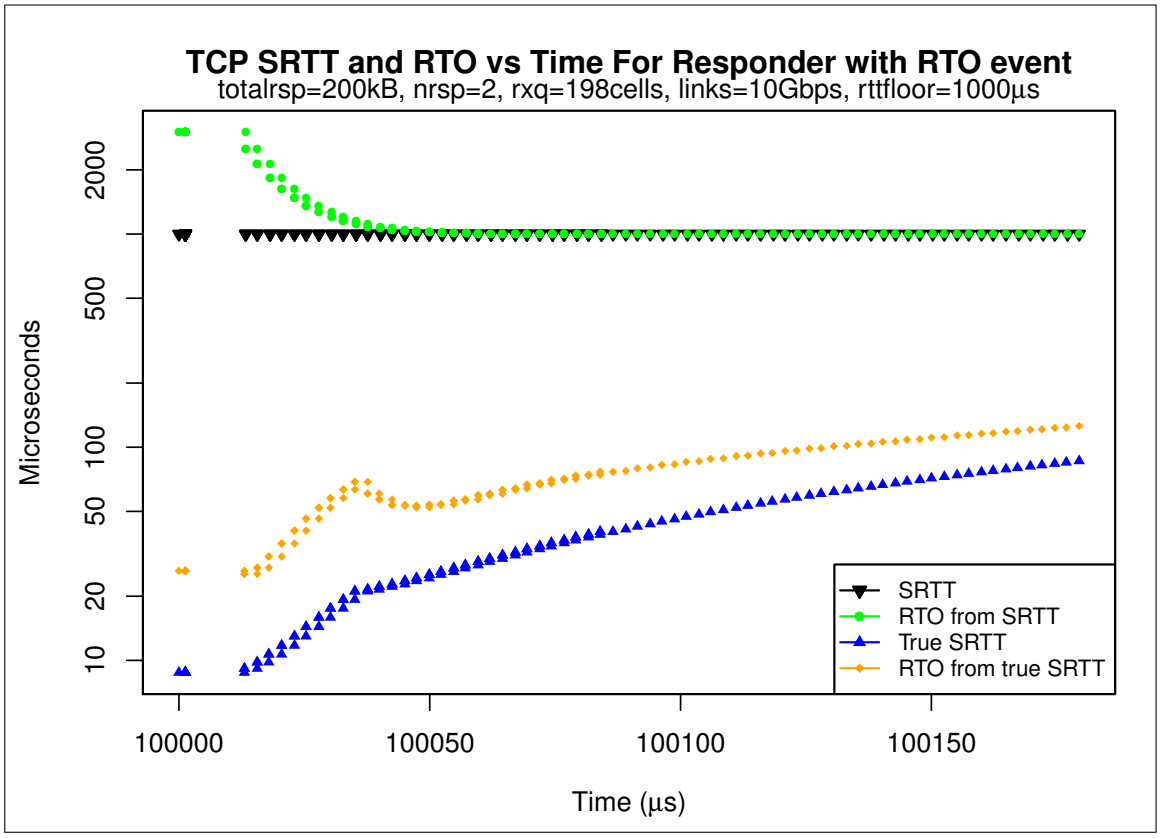


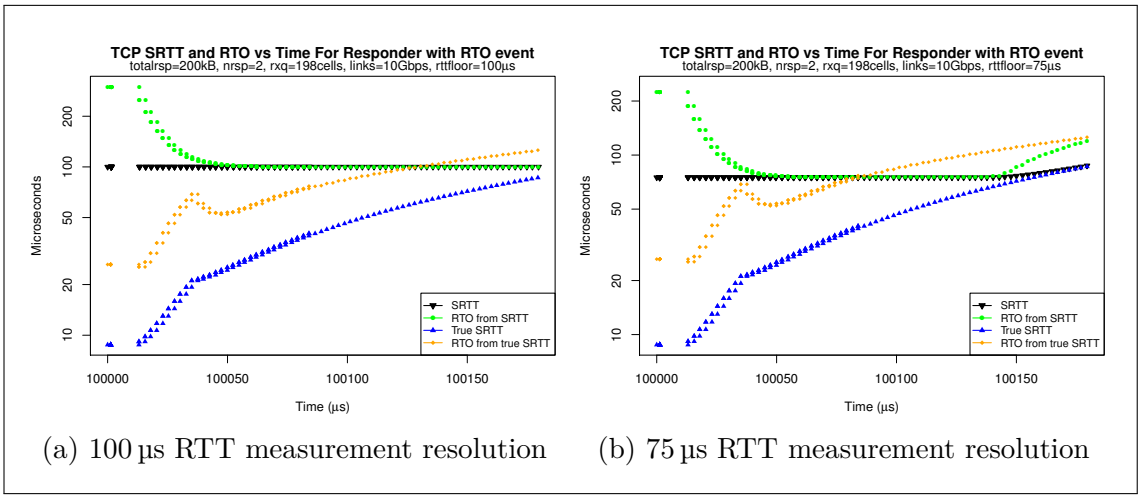
Figure 16: TCP SRTT and RTO versus time for the responder that experienced the RTO event.

TCP's millisecond resolution timestamps impose a 1 ms floor on the measured RTT, which by extension imposes the same floor on the RTO interval calculation. Figure 16 provides detailed insight into the relevant TCP stack variables (by way of SIFTR data) from the same experiment with a RTO shown in Figure 14. The measured and full resolution SRTT and corresponding derived RTO intervals are plotted up to the point in the experiment when the last cumulative partial ACK

was received at the responder i.e. when the retransmit timer was armed for the final time before the RTO.

Even though the true RTT experienced by TCP in this scenario is on the order of tens of microseconds, the measured SRTT and calculated RTO interval have insufficient resolution to adapt appropriately to the sub-millisecond latency path. The retransmit timer is therefore armed with a 1 ms interval instead of the appropriate 126 µs interval, which results in the grossly inflated completion time for the transaction seen in Figure 13.

If the path RTT were to continue increasing and approach the measurement resolution floor, there would exist a crossover period during which the RTO interval derived from the measured RTT would in fact be inappropriately *short*. This is because the measurement resolution floor effectively quantises RTT *variance* to zero, thereby nullifying the contribution of the $4 \times RTTVAR$ term in the RTO interval calculation. The RTO interval calculated for paths with *true* latency within the range $[rttfloor - 4 \times RTTVAR, rttfloor]$ will therefore be too short.



(a) 100 µs RTT measurement resolution (b) 75 µs RTT measurement resolution

Figure 17: TCP SRTT and RTO versus time for the responder that experienced the RTO event.

Figure 17 demonstrates this by comparing experiments with 100 µs and 75 µs RTT measurement resolution floors that were otherwise identicial to Figure 16. At 100 µs resolution shown in Figure 17a, the measured RTT still has insufficient resolution to even partially track the true RTT. The RTO interval derived from the measured RTT is therefore equal to the RTT on account of there being zero variance. The result is that the retransmit timer is armed with a shorter interval

than should be used.

However, had sufficient resolution been available to track the underlying changes to the true RTT, a non-zero variance would have been computed. That in turn would have armed the retransmit timer with the correct but longer interval. This is exactly what is observed in Figure 17b, where sufficient resolution is available to track the true RTT towards the end of the experiment. The RTO interval derived from the measured RTT increases as soon as change is observed.

This analysis makes for a cautionary note that using a RTT measurement resolution that is only slightly too coarse with respect to the path RTT can cause an inappropriately low RTO interval to be calculated accidentally. That in turn can lead to spurious retransmissions and incorrect detection of same.

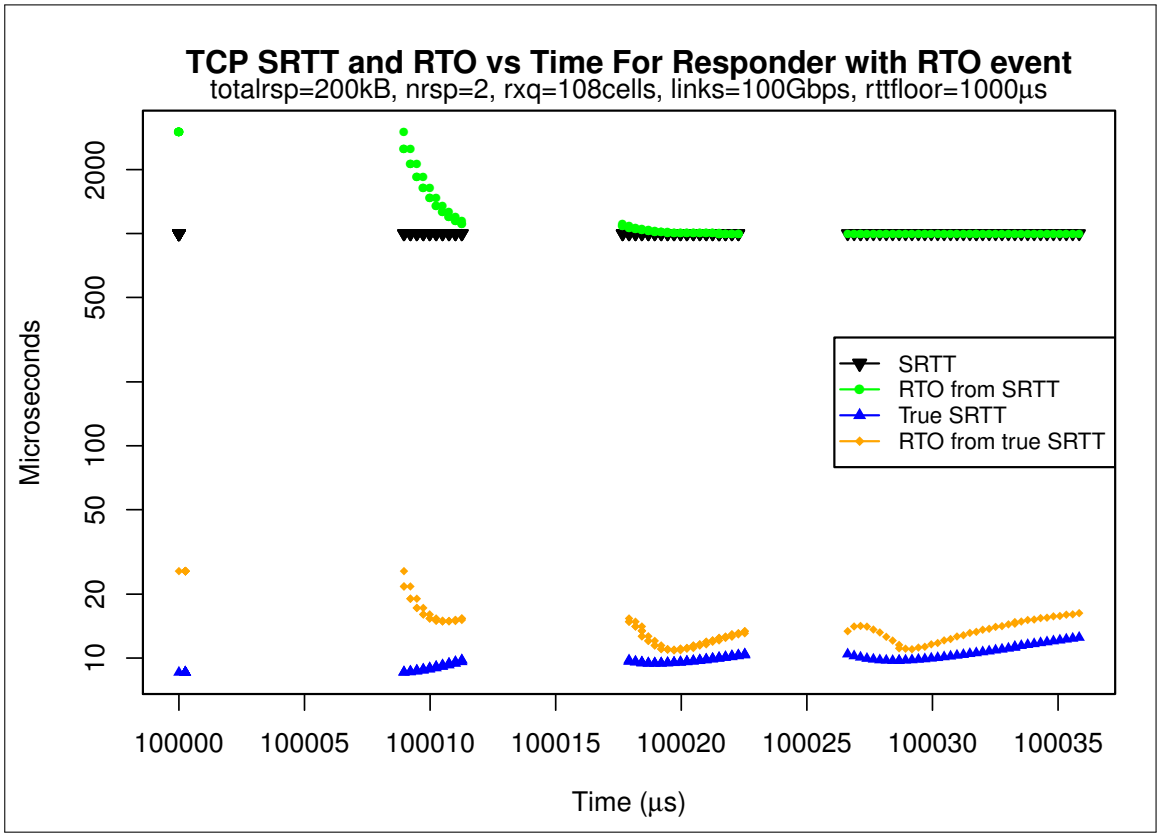


Figure 18: TCP SRTT and RTO versus time for the responder that experienced the RTO event, plotted up to the time of the last packet received before the timer expired.

If instead the gap between true and measured RTT widens on account of shorter path RTTs, the transmission opportunity loss and negative performance impact of a RTO will increase. To show this concretely, Figure 18 plots SRTT and RTO

responder TCP stack data for a single RTO event scenario similar to Figure 16, but with a 100 Gbps network instead of 10 Gbps. The gap between appropriate and used RTO intervals has increased to approximately 984 µs, or by approximately 110 µs relative to Figure 16. At 100 Gbps, a 984 µs RTO interval equates to 12.3 MB of lost transmission opportunity.

5.3.5 A Cautionary Note on Nagle's Algorithm

Figure 13 implicitly presents the $rxq = 199$ no loss experiment as a baseline against which to compare the other two experiments. Verifying how good a baseline it actually is provides an opportunity to briefly digress via a relevant point of discussion related to TCP-based transactional workloads.

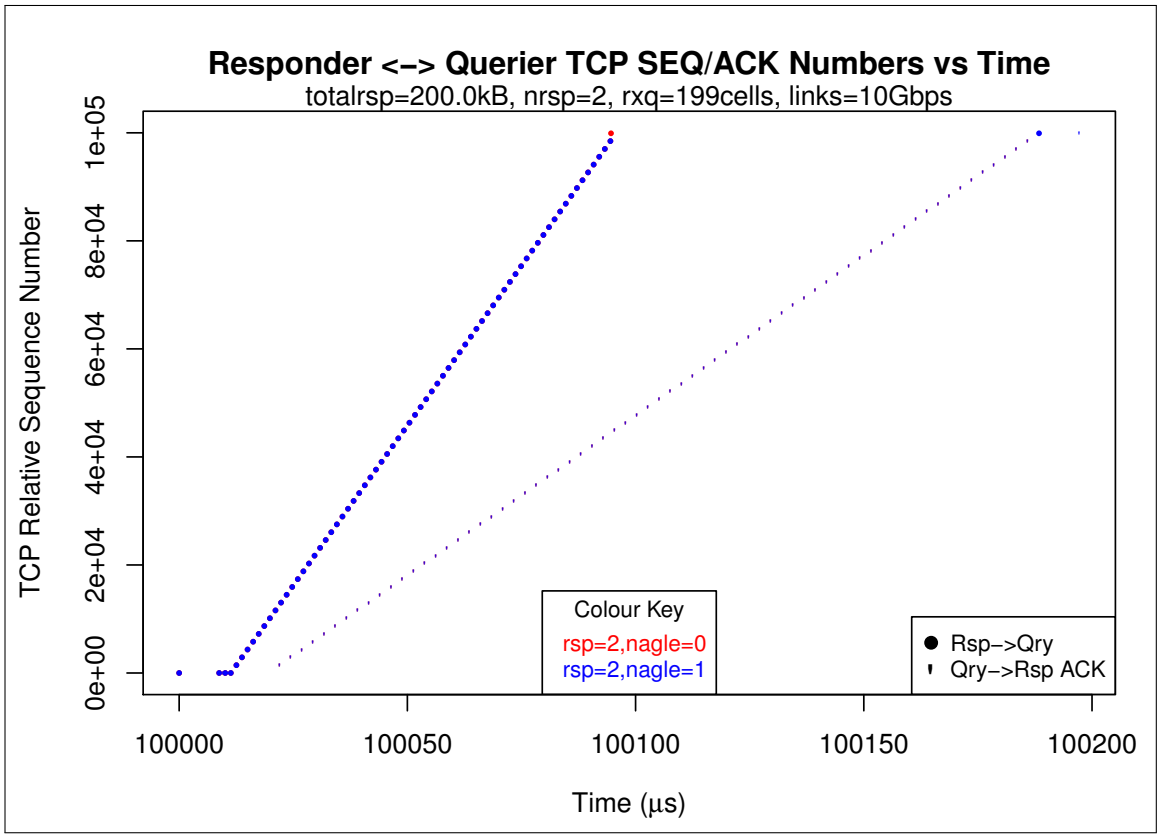


Figure 19: TCP sequence/acknowledgement numbers versus time, observed at the ingress switch port for the second responder. Nagle's algorithm delays the final response data packet by a RTT.

Figure 19 plots the evolution of the TCP sequence space for the baseline lossless experiment from Figure 13 as the "nagle=0" data series, confirming that the exper-

iment behaved as expected and is an appropriate baseline. However, this behaviour is non default for commonly used TCP stacks, which for historical reasons attempt to mitigate the so called "small-packet problem" [52] by employing what is commonly known as Nagle's algorithm. The "nagle=1" data series is from an otherwise identical experiment run with Nagle's algorithm enabled as a cautionary note.

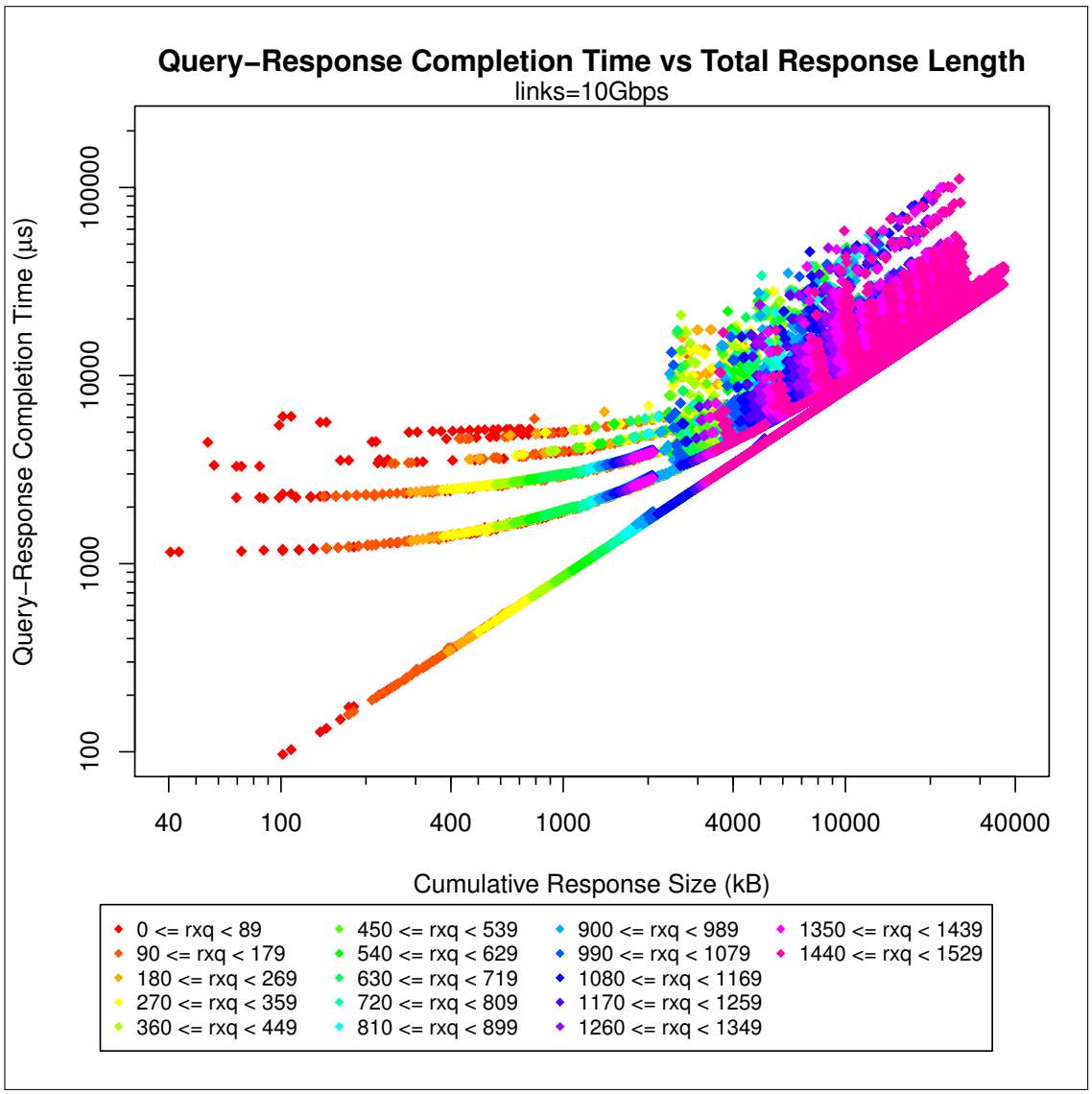
The point of interest with Nagle's algorithm enabled is that the final data packet sent by the responder is elicited by the querier's final ACK in that train rather than being sent together with the bulk of the response. This results in a transaction completion time that is inflated by an extra RTT, which includes the delay caused by the incast induced queue build up.

The 100 kB of response data returned by each responder is segmented into MSS chunks of 1448 B for encapsulation and transmission, which equates to 69 MSS segments and an 88 B remainder (a so called "small packet"). After the last MSS segment is ACK-clocked out of the responder, all subsequent received ACKs trigger the TCP send code which does not transmit the remainder on account of failing the Nagle check for in-flight data. Only on receipt of the final ACK in that train is *snd_una* updated to equal *snd_max*, which results in the calculation of zero in-flight data and the transmission of the remainder.

5.3.6 A Macro-level Look At Transaction Completion Times

Having explored key factors related to the incast phenomenon and its impact, it would be useful to gain some macro-level insight across a larger range of parameters. Figure 20 presents the query-response transaction completion times for a large set of experiments which varied the network speed, number of responders, per port VOQ buffer size and response length. Given the prohibitive number of permutations to produce a complete data set, larger than minimum variable step increments were used. The resulting set of permutations were then subjected to selection criteria and excluded if unlikely to be of interest. Variables took the following ranges, represented as [start, end, step]:

- rsplen: [start=14480 (initial_window), end=1035320, step=7240 ($5 \times MSS$)] and [start=13042 ($initial\_window - MSS + 10$), end=1035330, step=7240]
- nrsp: [start=2, end=35, step=3]
- rxq: [start=12, end=1524, step=18]

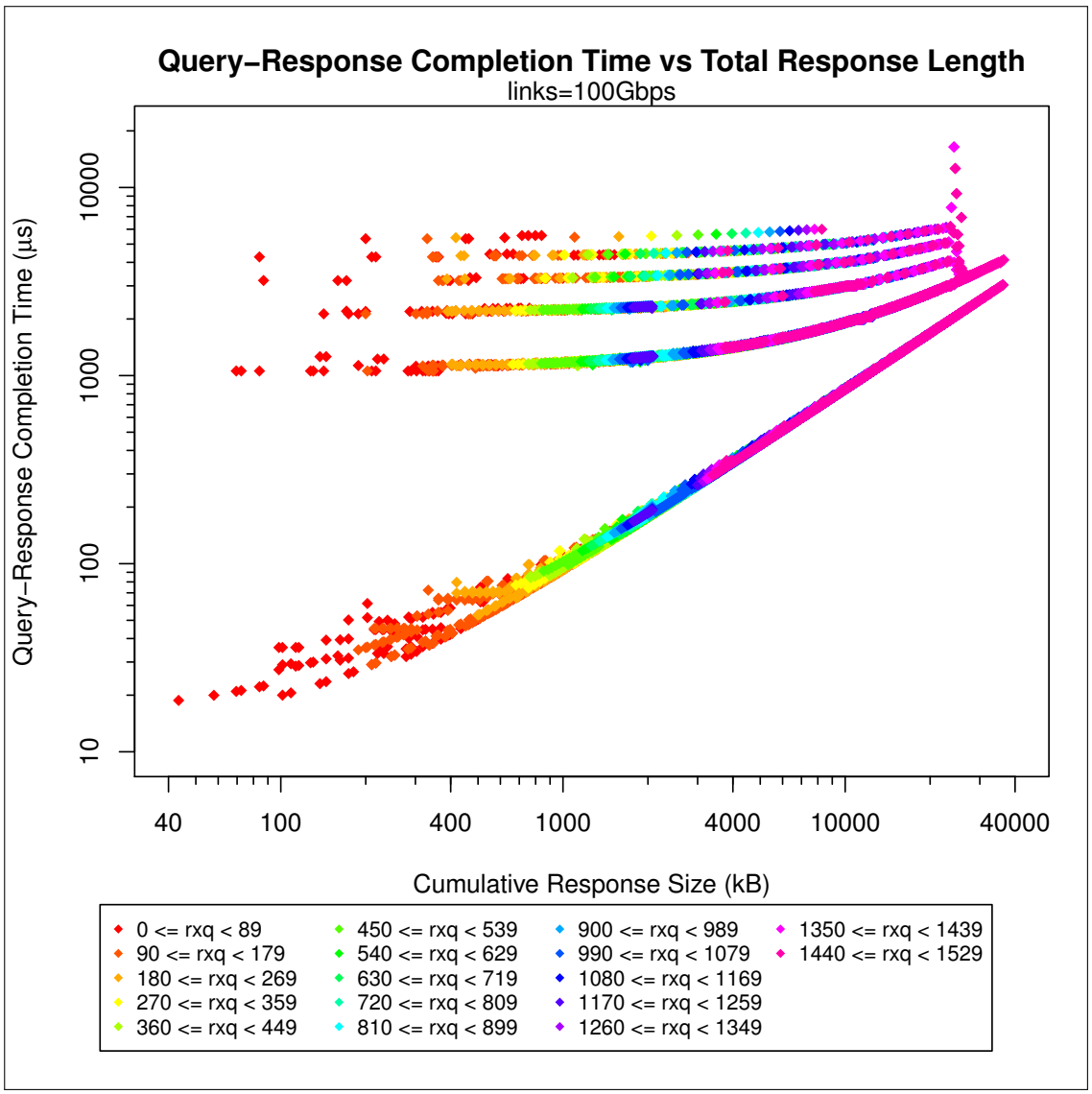


(a) 10 Gbps

Figure 20: Continued on next page.

- links: 10 Gbps and 100 Gbps

The range of response lengths covers transactions from tens of kB to tens of MB in size, representing transactions for small, medium and large objects (e.g., web response, sharded complex data, and sharded file on a storage cluster respectively). The second range probes non integer multiples of the MSS to provide visibility into any synchronisation effects that might exist with the VOQ buffer increment size. The range of responders was capped at a value similar to those typically discussed in existing literature [84, 6]. The VOQ buffer range with a cell size of



(b) 100 Gbps

Figure 20: Query-response transaction completion time versus total response length for variable number of responders and per port VOQ buffers.

256 B encompasses 2 to 254 frames per port. Although the numbers are less than the absolute buffer sizes found in many commodity data centre switches, the reasoning was two-fold: to study the pure incast phenomenon without interference from cross traffic or multiple transactions intrinsically requires less buffering, and investigating the relationship between the variables also does not require any particular amount of buffering. The range can be thought of as an exploration of buffer head room i.e. regardless of the absolute size of a switch's buffers, what happens when a transaction

takes place with a certain amount of available space.

The selection criteria were designed to probe behaviour just prior to queue saturation out to a reasonable way past queue saturation. To that end, a simplistic queue projection was applied to each permutation of the four variables and the permutation was rejected if any of the following tested true:

- The amount of response data returned by each responder fit completely within the amount of VOQ buffer less one full sized frame (permutations matching this criteria would all produce baseline lossless results and therefore contribute minimal new information to the data set).
- The projected queue high water mark was less than the amount of VOQ buffer less one full sized frame (similarly, permutations matching this criteria would all produce baseline lossless results and therefore contribute minimal new information to the data set).
- The projected queue high water mark was greater than twice the amount of VOQ buffer less one full sized frame (permutations matching this criteria would overrun the queue by a substantial enough margin that the amount of loss and associated impact on the transaction would render the experiment so unlikely as to be of lesser interest).

The queue projection was conservative in its modelling and overestimated the high water mark by a consistent margin to ensure the first two criteria did not inadvertently reject experiments that were on the borderline of saturating the queue. Of the total 581400 possible permutations, 426671 were excluded by the selection criteria, resulting in a data set consisting of 154729 experiments.

Baseline completion times corresponding with lossless experiments form a lower bound y-axis floor. The floor is primarily a function of the serialisation time for the total response length, which manifests as the baseline exhibiting a positive gradient with respect to total response length.

Four distinct tiers corresponding with the number of non-parallel RTO events incurred during each experiment are clearly visible, each separated by the 1 ms RTO floor. The larger gap between baseline and RTO experienced completion times at 100 Gbps account for the more distinct tiers in Figure 20b. Noise visible around the baseline and each RTO tier is latency associated with losses being repaired by one or more fast recovery episodes.

Perhaps the most important observation relevant to the remainder of this thesis is that there are numerous scenarios in which transactions can incur one or more RTO events. As previously discussed, each RTO stalls the connection for significantly longer than it would if RTT could be measured with sufficient resolution. Each non-parallel RTO therefore compounds the lost transmission opportunity resulting from the inappropriate RTO interval.

5.3.7 Transaction Completion Time Versus RTT Measurement Resolution

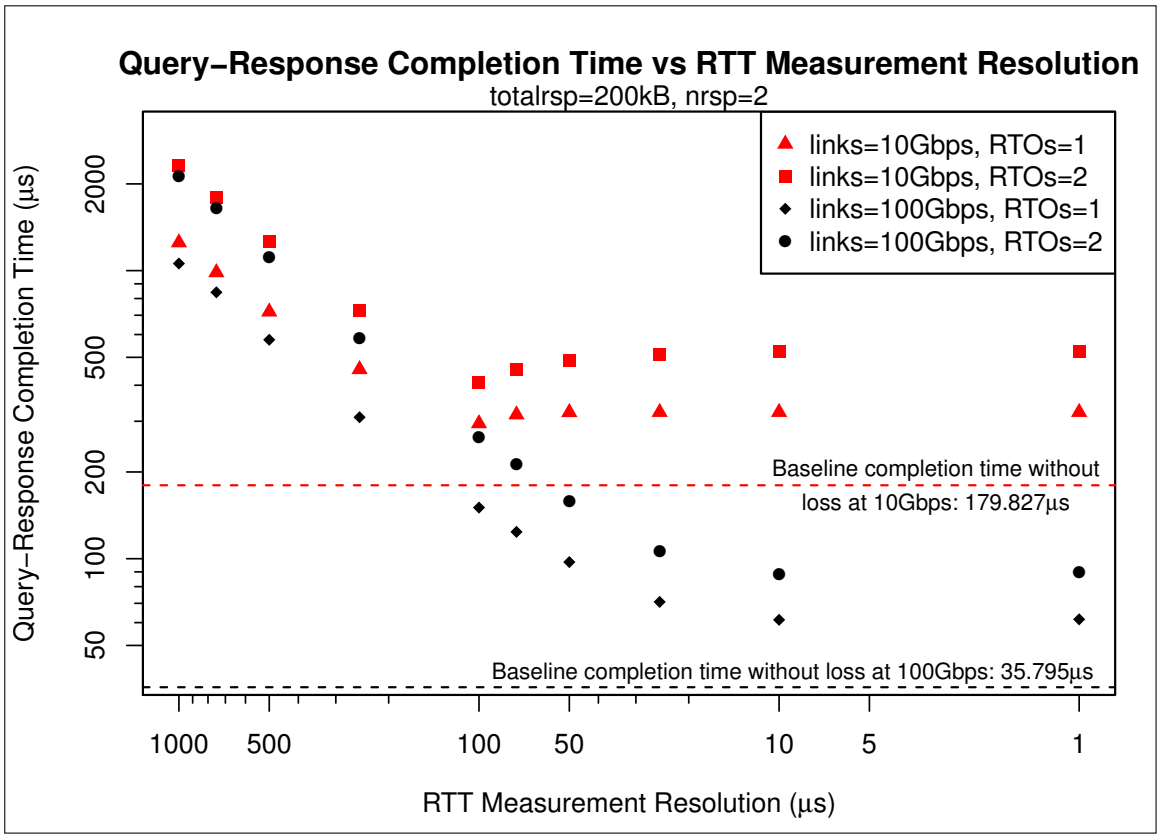


Figure 21: Query-Response completion time versus RTT measurement floor.

So how much improvement would be possible for flows experiencing RTO events if a single change was made – namely to increase the RTT measurement resolution and thereby lower the calculated RTO interval? Figure 21 presents transaction completion time as a function of RTT measurement resolution for single and double non-parallel RTO scenarios at both 10 Gbps and 100 Gbps.

The gap in completion time between the baseline completion time line and single

RTO event data points is equal to the RTO interval. The left points at 1000 µs resolution correspond with the existing 1 ms resolution in use today. As RTT resolution increases and the corresponding RTO interval is able to shrink to more appropriate values, transaction completion times improve substantially. Moving from a 1 ms to 1 µs resolution improves the transaction completion times for the single RTO scenarios from 1251.0 µs to 322.5 µs at 10 Gbps and 1058.2 µs to 61.6 µs at 100 Gbps. The same amount of improvement is observed for dual RTO scenarios as expected. The absolute numbers and relative improvement vary predominantly with network speed and total response length, but note that only speed is varied in Figure 21.

The counter-intuitive increase in completion times for the <100 µs 10 Gbps data points relative to the 100 µs points is the practical consequence of the earlier discussion surrounding Figure 17. It is not particularly discernable from the plot, but a minor increase also exists between the 100 Gbps 10 µs and 1 µs data points. The shorter completion times correspond with experiments that resulted in mistakenly short RTO intervals being calculated because the path latency fell within the $[rttfloor - 4 \times RTTVAR, rttfloor]$ range.

These findings make a strong case for pursuing improvements to TCP's RTT measurement machinery to reduce the damage sustained by RTO events.

6 Adapting TCP's Control System for the Data Centre

The CLUES-based TCP incast investigation leads us to the final contribution of this thesis - defining and characterising a mechanism which minimises the impact of incast events and complements other incast mitigation mechanisms.

The review of existing literature revealed a wide range of incast mitigation proposals, each varied in their approach and point of insertion in the end-to-end communication path between applications. As no practical scheme can utilise perfect knowledge about the end-to-end path, we focus on adapting the transport protocol to the realities of best-effort IP networks operating at data centre speeds and latencies. To that end, we mitigate the most undesirable effects of incast by designing and implementing an adaptive, higher-resolution TCP control system that ultimately improves responsiveness and goodput.

As described in 2.4, TCP's control system is comprised of dynamic interactions between explicit signaling, inferred feedback, timers and finite state machine transitions. Measured RTT is a fundamental control input affecting many of these interactions either directly or indirectly. The error between true and measured RTT therefore has critical implications for TCP in general, and particularly in the data centre.

We know from the incast chapter that the loss patterns typically associated with TCP incast congestion events can require repair via the RTO timer. It therefore stands to reason that if we cannot eliminate RTO events, we should try to minimise and bound their impact as much as possible.

6.1 Revisiting TCP's RTT Measurement Machinery

Section 2.4.3 motivates our focus on improving TCP's RTT measurement machinery for data centres. By equipping TCP with the ability to accurately measure a significantly wider spectrum of path latencies, we can improve both worst case and variability of performance. An additional benefit is that for low-latency paths, measured RTT becomes a useful control input for non-core protocol mechanisms like delay-based congestion control. Such mechanisms have the potential to yield even greater performance and/or dynamic behaviour improvements.

Aspects of the RTT measurement machinery were reimagined in such a way as

to maximise reuse without violating the various constraints codified in existing TCP implementations. The intent was to yield a minimal modification that would be both interoperable and incrementally deployable. The wide availability of the timestamp option in TCP stacks coupled with its workable constraints and established use for RTT measurement made it a design focal point.

6.1.1 Design Goals and Considerations

Before embarking on a discussion of possibilities, we must establish any constraints and their degree of inviolability within the context of the stated goals and any other relevant considerations.

Maintain comparable robustness

The PAWS mechanism, as specified and widely implemented, increases TCP's robustness by reducing the opportunity for stream corruption. The MSL in concert with the interplay between rate of increment of sequence numbers versus TSval values affect the robustness improvement afforded by PAWS.

There is no clear cut guidance on bounds for MSL, but it is intuitively obvious that a network's true MSL is directly tied to the probability of stream corruption. Consideration of MSL is therefore really about probability and risk management rather than any sort of absolute correctness.

The MSL upper bound of 255 s referred to in RFC 1323 is based on the long-irrelevant idea that the 8-bit IP Time To Live (TTL) field represents a value in units of seconds. The field was to be decremented for each second the packet remained in the network i.e. an IP packet should never (to a first approximation given that propagation delays were likely ignored) arrive at its intended destination more than 255 s after it was sent.

In practice, IP routers decrement the TTL field by one on forwarding a packet, without any concern for the packet's length of stay i.e. there is no formally enforced hard limit on the length of time a packet can remain actively forwarded within an IP network.

Other choices for an MSL upper bound are similarly heuristic in nature. RFC 793 explicitly states that the 2 min MSL value referred to in the document was arbitrarily chosen. The implementation reality is that operating systems often allow operators to directly or indirectly set the notional TCP MSL. This provides a means

to control the length of time connections linger in the `TIME_WAIT` state tying up system resources. For some concrete numbers, both FreeBSD 11.0 and Linux 4.6 default to a 30 s TCP MSL.

The conclusion to be drawn is that MSL considerations do not impose any strong requirements on potential RTT measurement schemes. Schemes which provide PAWS protection over longer MSL upper bounds would impart more robustness and therefore be more desirable. Schemes which afford the operator some level of control over the effective MSL interval, and consequently their appetite for risk of stream corruption, would be more desirable still.

There are two primary robustness related considerations regarding the interplay between rate of increment of sequence numbers versus TSval values. The TSval field should not:

- increment too slowly such that the sequence number can wrap within an increment interval
- increment too quickly such that the TSval field value and sequence number can wrap within a similar period of time that is less than the notional MSL interval

Both cases would render a receiver's PAWS checks helpless to prevent stream corruption.

Interoperability

Interoperability imposes a requirement not to change the TCP header or timestamp option wire format. It also requires that any changes to the TSval field semantics do not materially alter an unmodified receiver's processing of segments from a modified sender. The latter hinges on RFC 1323's PAWS related pronouncements and their associated implementation in conformant TCP stacks. Specifically, the function or mechanism used to derive TSval field values must allow receivers to determine relative segment order (i.e. maintain the RFC 1323 requirement that the field never goes backwards except when wrapping). It must also be compatible with the PAWS idle time failure mode mitigation logic (or if incompatible, avoid triggering the failure mode altogether).

Adapt RTT measurement resolution to path RTT

Schemes should be capable of measurement resolutions that encompass the full spectrum of current and future data centre path latencies. They should also ideally be capable of measuring wide area path latencies to avoid stacks having to maintain multiple schemes. Current end-to-end IP paths potentially consist of a diverse mix of link layer hop technologies (e.g., satellite, 100 Gbps Ethernet and virtualisation technology emulating network links directly across a system's backplane). The range of encounterable path latencies therefore ranges from seconds down to nanoseconds. At the low end of the range typical in data centres, commodity switches with 350 ns port-to-port forwarding latency are available today [263].

The speed of light in a vacuum (29.98 cm/ns) sets the lower bound limiting factor on path latency[10]. Research advances in photonics hold the promise of practical latencies approaching the theoretical lower bound in the near future. Air-filled hollow fibre optic cable capable of 29.00 cm/ns propagation velocity [264], replacing metal with optical interconnects between and within ICs [265], and optical switching [266] have been demonstrated. In practical terms, these advances translate into lower bound latencies between physically separate computers on the order of tens of nanoseconds. Intra-computer latencies across backplanes or within individual ICs could be in the picosecond range over sufficiently short communication channel distances. Resolution adaptation down to the order of picoseconds would therefore seem like a desirable lower range bound for the future.

Given the range of possible resolutions, it would be desirable for schemes to adapt resolution measurement per connection or group of connections with similar path RTT. This would allow overheads associated with higher resolution measurement to be restricted to connections that actually need it.

Sufficient resolution for variance measurement

As discussed in 5.3.4, the RTT measurement resolution needs to be sufficient to capture variance. If it is not, an inappropriately low RTO interval may be calculated accidentally which can lead to spurious retransmissions and incorrect detection of same. This has clock choice implications for any given connection and its particular path latency.

[10] Until a superluminal communication mechanism is discovered and the physics books rewritten!

Other considerations

Hardware acceleration technologies, including the widely used TCP Segmentation Offload (TSO)/Large Receive Offload (LRO), are a relevant concern in the data centre ecosystem. NICs with the TSO/LRO capabilities allow the network stack and NIC to exchange pseudo packets which are larger than the MSS.

With TSO, the large pseudo packet to be transmitted is passed to the NIC together with a template TCP/IP header. The NIC's hardware then divides the data into MSS segments, attaches the template header, updates fields like sequence number and checksum as required, and finally transmits the properly formed segments. LRO is complementary, in that the NIC will merge segments received on the wire into a pseudo packet. The pseudo packet is then flushed, based on some typically configurable criteria, up the stack for processing by protocol layers. Used together or in isolation, TSO and LRO can improve a system's overall per-packet processing efficiency. This gain does come at the expense of some protocol layer flexibility and additional but often configurable latency.

There are some relevant technical details that a RTT measurement scheme reliant on the timestamp option should be robust in the face of. TSO implementations are intentionally "dumb" with respect to TCP options, and typically copy the options component of the template header verbatim into each segment. TCP connections utilising the timestamp option and TSO therefore do have bursts of packets leaving the sender with identical timestamp option data. By contrast, the intelligence of LRO implementations is more varied, with many being aware of the timestamp option. Such implementations may update the pseudo packet header's timestamp option data with that of the most recently received segment being merged into the pseudo packet.

6.2 ARREAR: An Adaptive Resolution RTT Measurement Scheme for TCP

The Adaptive Resolution RTT mEAsuRement (ARREAR) scheme is a sender-side modification of TCP's timestamp-based RTT measurement mechanism. The scheme's genesis can be traced to an observation made in RFC 1323 that a receiver treats TSval as a serial number. If a sender decouples TSval field increment rate from real time, thus treating it like a serial number, adaptive resolution RTT measurements can be made in a way that respects our design goals and considerations.

Note that ARREAR is one plausible possibility selected from the TSval-as-a-serial-number solution space for implementation and evaluation.

6.2.1 Details

Since TSval can be an arbitrary value, so long as it satisifes the previously discussed requirements, it can therefore be used to transmit a token which only has meaning to the sender. Conceptually, ARREAR treats the 32-bit TSval space as a pool of measurement tokens, with each increment of TSval to be made when a new RTT measurement is desired. Senders cache the TSval "token" value paired with a clock read, and when it returns as a TSecr, take a measurement by subtracting the cached clock read from the current time. Measurement resolution is therefore tied to the quality of the clocks a sender has access to, and the possibility exists to use different clocks for different connections.

It follows that the frequency with which measurements are taken and the TSval wrap period become elastic at the sender's discretion, which imparts useful flexibility. The ability to perform RFC 1323-style per data-acknowledgement pair measurements remains if a sender increments TSval for each segment. Why not instead couple measurement rate to a path's RTT and the variance thereof? A sender could, for example, set the Nyquist rate as the lower bound and choose the measurement rate based on RTT variance.

ARREAR retains the TSval field semantics required for PAWS to be effective. TSval never goes backwards except at wrap, which allows a receiver to determine relative segment order using the sequence number and TSval. The interplay between sequence number and TSval rate of increment is now explicitly controlled by senders, which can ensure the TSval does not increment too fast or slow relative to the sequence number. If the sequence number wrapped since the last RTT measurement, the sender can detect this and increment TSval even if a measurement would not have otherwise been attempted for the next segment. Even with low path latencies and/or high RTT measurement rates, ARREAR avoids triggering the PAWS idle time failure mode at receivers. The TSval field only increments as segments are being sent, and therefore the delta between an incoming TSval and receiver's TS.Recent should never exceed 2^{31} regardless of elapsed idle time.

With the TSval increment frequency (and therefore wrap period) decoupled from real time, ARREAR allows connections to individually control the PAWS protection interval. The TSval increment (i.e. RTT measurement) rate can be manipulated to

ensure that the sender would never reuse a particular combination of sequence number and TSval value within the desired MSL period. For example, dividing the token pool (2^{32}) by the notional MSL and multiplying the result by the RTT yields the per-RTT token consumption rate that would avoid wrapping the TSval field within the MSL period. This allows an implementation to offer operator/application level control of risk tolerance for stream corruption from duplicate segments by changing the TCP MSL. Many application layer protocols perform their own checksumming despite TCP's mostly reliable delivery. Conceivable value therefore exists in having the ability to trade off MSL period against RTT measurement rate on a per system or socket basis.

The scheme's biggest drawback is its reliance on additional sender-side state to maintain mappings between TSval values and clock reads. Being a function of RTT and measurement rate, the number of mappings increases in line with both. The simplest implementation conceivable is a FIFO queue of in-progress measurement mappings (which is precisely what we implemented and evaluated). Various ways to reduce the computational overhead and amount of state required have been considered and are outlined as future work, but this thesis' focus is characterising the baseline scheme.

6.2.2 Implementation Specifics

The implementation of ARREAR completed in support of this thesis uses the `sbinuptime()` kernel function as its clock source. In CLUES, this function procures its data from ns-3's scheduler, which provides a 1 ns resolution by default. The `sbintime_t` data type returned by `sbinuptime()` is used by ARREAR and the supporting TCP stack changes for time storage.

The new `tcp_arrear_gentsval()` function generates the TSval to write into a segment's header. The function either returns the previous TSval token or a new one depending on the decision to attempt a measurement or not. Each measurement attempt stores the paired TSval token and `sbintime_t` clock read in a FIFO queue of pending measurements in the connection's control block.

Deciding whether to attempt a measurement or not depends on a few factors. The TCP output path passes a boolean indicating whether or not the stack expects the outbound segment would elicit an immediate response from the peer (e.g., has the SYN flag set) and therefore make a good measurement candidate. If the stack does not consider the outbound segment to be a candidate, the current ARREAR

implementation will not attempt a measurement and returns the most recently used TSval.

If the outbound segment is a measurement candidate, the ARREAR implementation will attempt a measurement subject to a token rate limiting calculation. This rate limiting mechanism is the means by which the implementation ensures the TSval field does not wrap inside of the configured MSL period. Approximately once per RTT, the token use rate is compared against the use rate necessary to meet the MSL wrap period. If the current use rate is too high, an appropriate rate limit is set in the form of a duty cycle i.e. for every X attempted measurements, the next Y measurements which would have otherwise been attempted should be skipped. The decision and parameters with which to rate limit are reviewed every RTT and adjusted as required.

Each returning ACK with a TSecr token calls the new `tcp_arrear_pair()` function from the TCP input processing path. The function walks the pending measurement queue starting at the oldest entry, removing any entry it encounters with a token prior to the received TSecr token. It stops on finding a match, or on finding a larger token which implies the incoming token is not a measurement candidate. If a match is found, the associated clock read is subtracted from the current time and the resulting RTT measurement returned to be fed into the SRTT and RTO update logic.

Removing pending measurements from the queue which are logically before the incoming TSecr amortises queue maintenance cost across all ACKs and keeps the queue at a minimal size. This simplifies the implementation in the face of ACK loss and/or LRO coalescing leaving orphaned pending measurement entries in the queue. A caveat is that ACK reordering would lead to measurements being prematurely abandoned that otherwise would have completed successfully.

A more sophisticated implementation could easily deal with this shortcoming by adding a TTL mechanism to pending measurements. As return path reordering has not been noted as a prominent concern in data centre networks (to the best of my knowledge) and is not present in CLUES (unless explicitly configured), the simple implementation was preferred.

6.3 Evaluation of ARREAR

The methodology used to evaluate ARREAR was essentially identical to that documented in 5.2, with the primary difference being that ARREAR was enabled by

way of a sysctl control knob added to the CLUES stack.

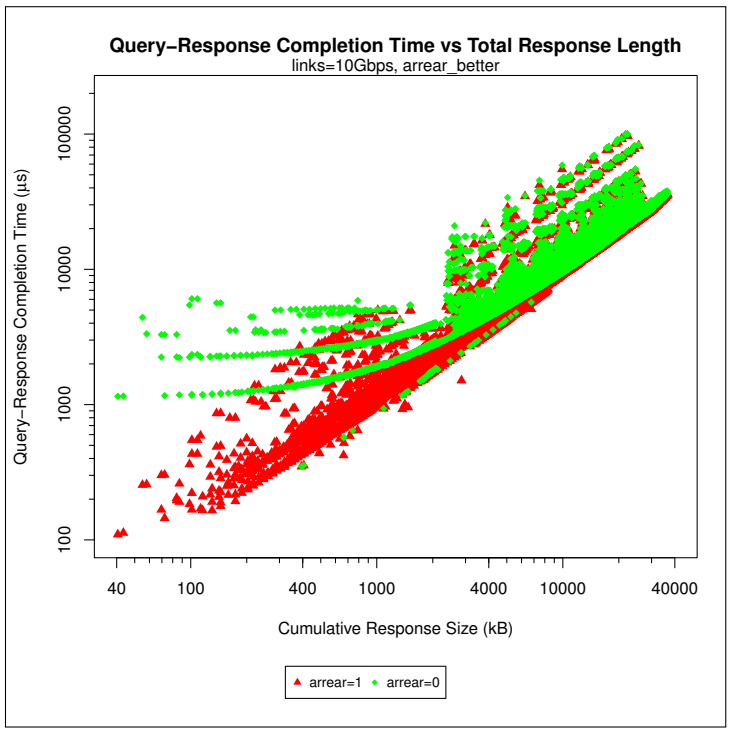
Let us begin with a macro level comparison of ARREAR to follow on from the end of Chapter 5. For every experiment presented in Chapter 5 Figure 20, an experiment with ARREAR enabled but otherwise identical was also performed to produce a sister data set. Figures 22 and 23 present this sister data set plotted against the ARREAR-disabled data set from Figure 20.

With the 1 ns resolution clock source provided by CLUES, ARREAR had an order of magnitude finer resolution than was required to accurately measure end-to-end RTT for all scenarios in the data set. The expectation is therefore that the measured RTT and derived RTO would correctly adapt to the true RTTs experienced during each experiment. This in turn should be reflected in the data as an identical baseline but less distinct RTO tiers compared with Figure 20, which is realised in Figures 22 and 23. Completion time deviation above the baseline consists of a mix of high resolution RTO events and/or fast recovery episodes, the approximate proportion of which is no longer easy to visually discern.

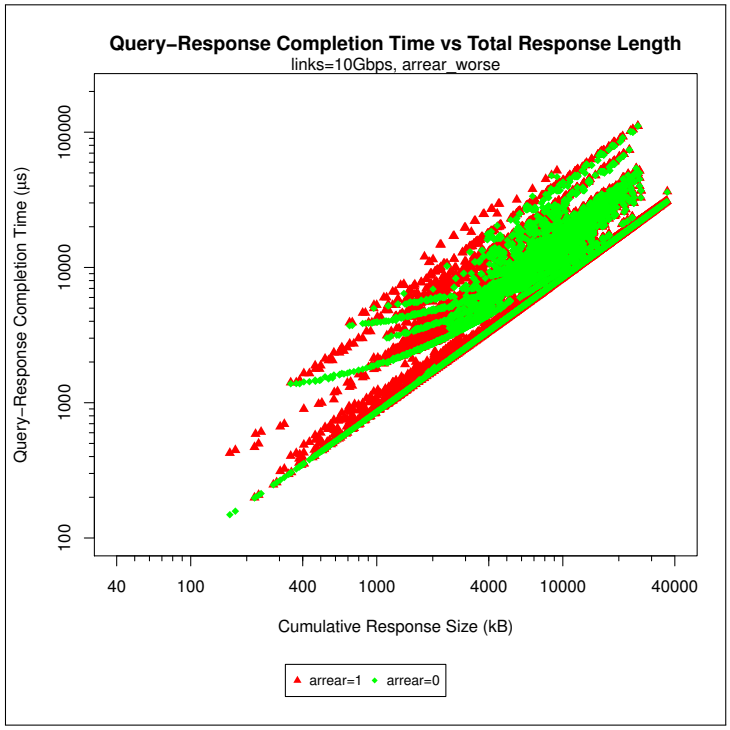
With ARREAR enabled, any inadequacy in the standard TCP algorithms and formulae pertaining to RTT have the potential to be exposed and possibly exacerbated by the data centre environment. Spurious retransmits triggered by inappropriately low RTO intervals become a potential source of problems. They cause congestion window collapse and premature transition to congestion avoidance mode, thereby slowing the latter part of the transaction. This is the reason why a subset of experiments take longer to complete with ARREAR than without. Initialisation and calculation of SRTT/RTTVAR/RTO per RFC 6298 can clearly have undesirable effects in some incast scenarios. This warrants future work to explore other potentially more appropriate algorithms.

Of the total 154729 experiment pairs, Figures 22a and 23a plot the 27207 10 Gbps and 29920 100 Gbps pairs respectively that recorded shorter transaction completion times with ARREAR. Figures 22b and 23b plot the 45224 10 Gbps and 1328 100 Gbps pairs respectively that recorded longer transaction completion times with ARREAR. The 5652 10 Gbps and 45398 100 Gbps pairs that recorded identical completion times were excluded from the plots.

ARREAR has the expected and desired effect of vertically compressing the ARREAR-disabled data set for the experiments shown in Figures 22a and 23a. Completion times for experiments that experience one or more non-parallel RTO events are significantly reduced. Where previously a multiple of the RTT measurement res-

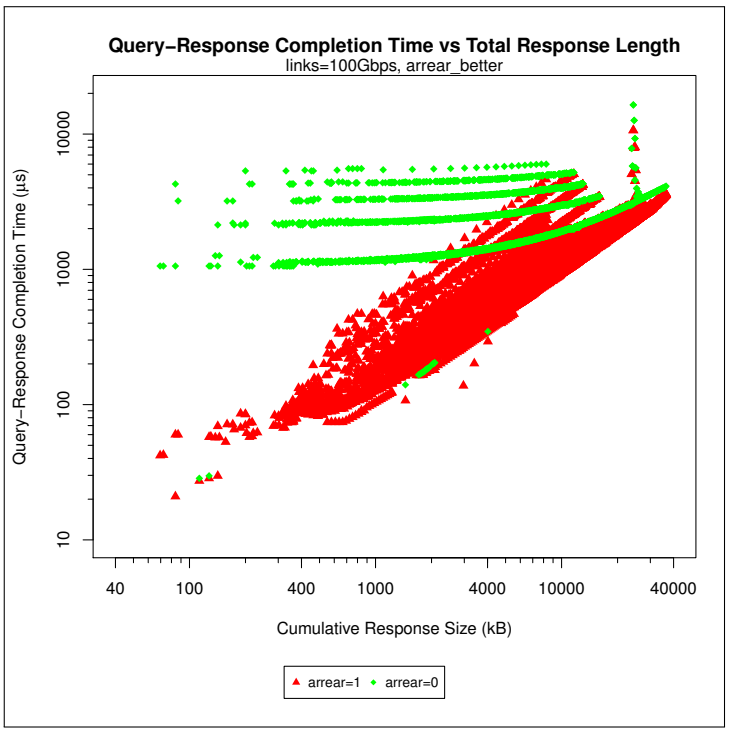


(a) Experiments with ARREAR enabled that completed in less time.

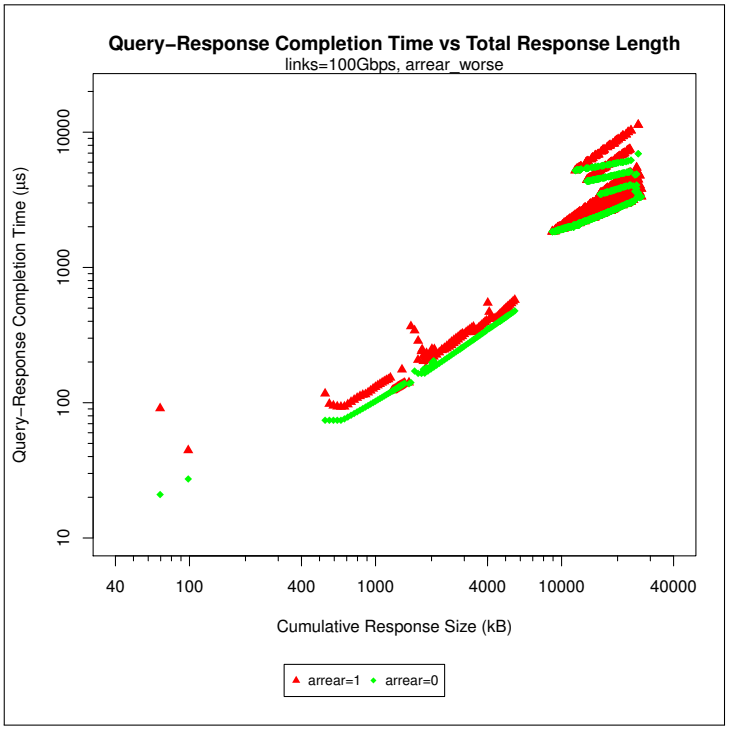


(b) Experiments with ARREAR enabled that completed in more time.

Figure 22: Transaction completion time versus total response length at 10 Gbps.



(a) Experiments with ARREAR enabled that completed in less time.



(b) Experiments with ARREAR enabled that completed in more time.

Figure 23: Transaction completion time versus total response length at 100 Gbps.

olution (1 ms without ARREAR), with ARREAR they become a multiple of each experiment's unique RTO interval. As already discussed, the amount of transmission opportunity lost with each RTO increases with increasing network speed and/or decreasing path latency. This fact is also clearly borne out by the larger relative improvement in completion times observed in Figure 23a as compared with the 10 Gbps experiments in Figure 22a.

Figures 22b and 23b compare the subset of experiments that recorded longer transaction completion times with ARREAR. Recall from 5.2 that the CLUES stack was modified to remove RTO_{min}, the fixed lower bound applied to RTO interval calculation. This was done to ensure the calculation became completely adaptive to RTT measurement resolution, thereby providing experimental insight into the effects.

6.3.1 Reducing Transaction Completion Time

TCP time sequence diagrams help to visualise the underlying transport dynamics which led to the improvement in transaction completion times with ARREAR. Figures 24a and 24b show two representative pairs from Figure 22a, with one[11] and two[12] non-parallel RTO events respectively.

The only point of difference between both pairs is the reduced RTO interval derived from the high resolution RTT measurements provided by ARREAR. This directly translates into reduced application layer latency and therefore a corresponding reduction in the transaction completion time.

6.3.2 Spurious Retransmits

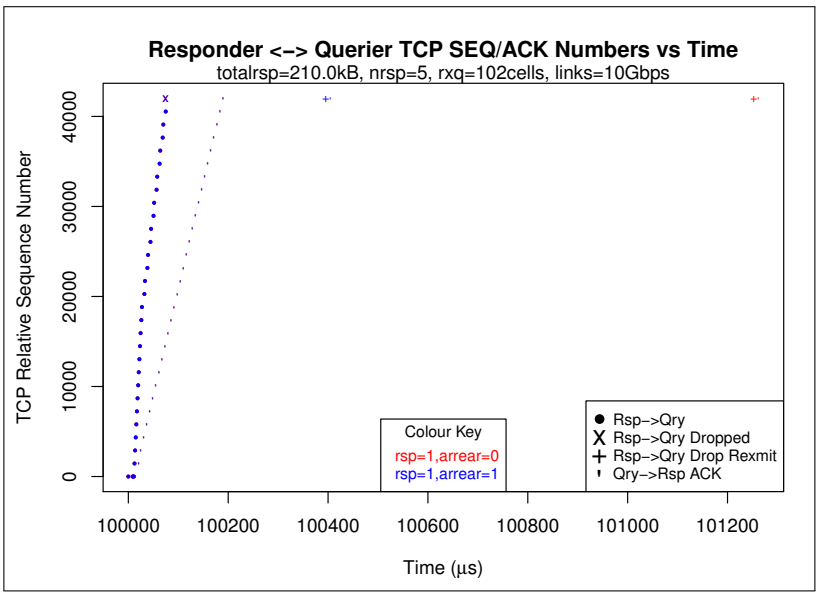
Of particular interest are the 30.1% of experiment pairs which exhibit longer completion times with ARREAR than without. Figures 25a and 25b are TCP time sequence diagrams created for pairs of ARREAR enabled/disabled experiments that exhibited the transaction completion time inversion.

Comparing ARREAR enabled and disabled data series in Figure 25a[13] reveals a spurious retransmission occurred approximately 40 µs into the ARREAR-enabled

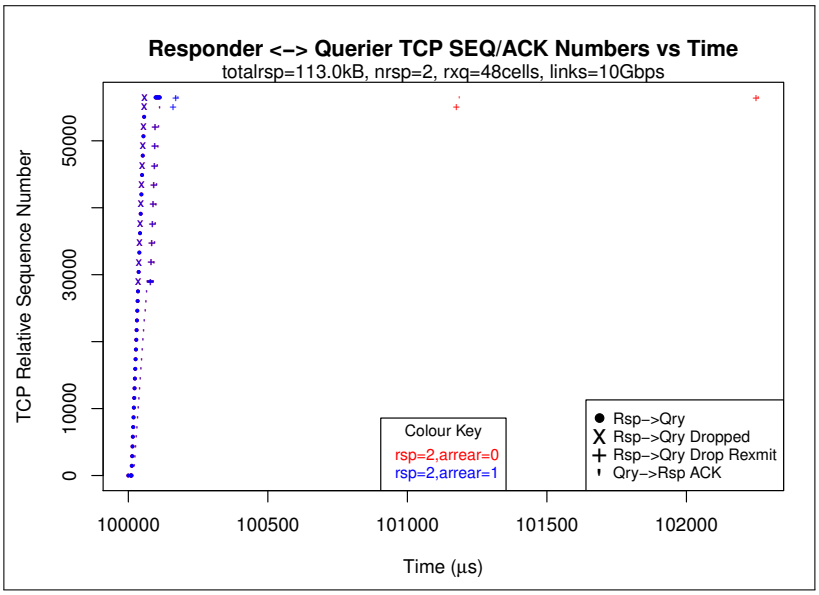
[11] Experiment utilised 5 responders returning a 210.0 kB cumulative response over a 10 Gbps network with 102 cells of VOQ buffering per port.

[12] Experiment utilised 2 responders returning a 113.0 kB cumulative response over a 10 Gbps network with 48 cells of VOQ buffering per port.

[13] Experiment utilised 8 responders returning a 278.1 kB cumulative response over a 10 Gbps network with 120 cells of VOQ buffering per port.



(a) 1 RTO.

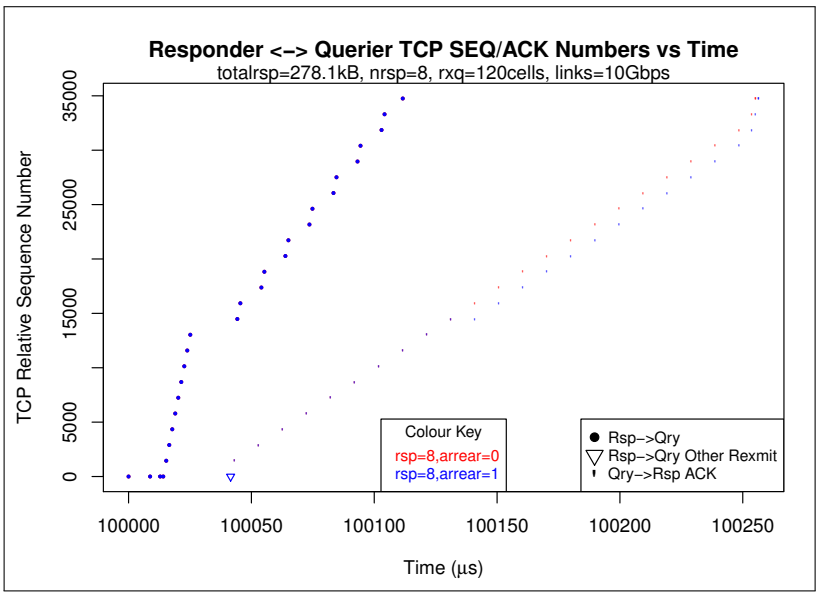


(b) 2 RTOs.

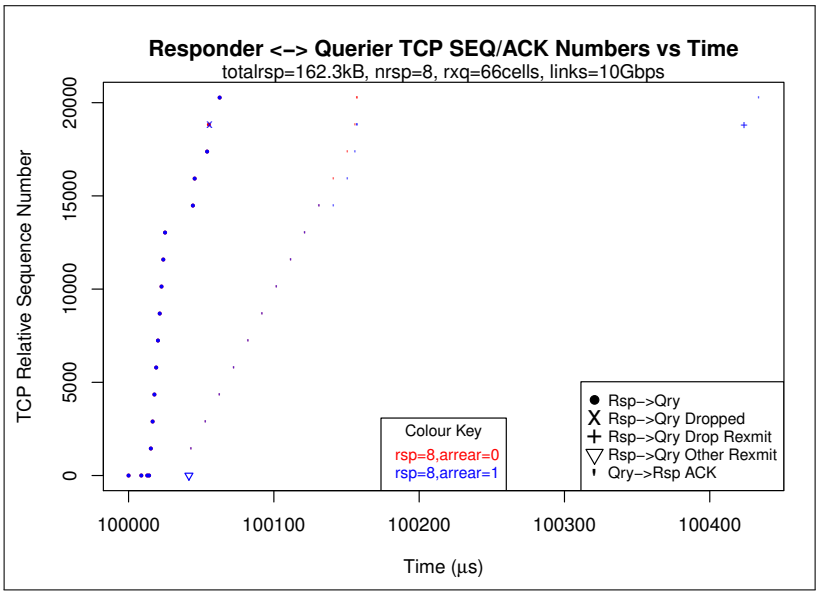
Figure 24: TCP sequence/acknowledgement numbers versus time, observed at the ingress switch port of the labelled responder.

experiment.

TCP connections only get to make a single RTT measurement during the connection establishment handshake before putting an initial window's worth of data on the wire. Per RFC 6298 [262], the first RTT measurement results in the stack setting $SRTT = RTT$, $RTTVAR = \frac{RTT}{2}$ and $RTO = SRTT + 4 \times RTTVAR$ i.e.



(a) 1 spurious RTO.



(b) 1 spurious, 1 non-spurious RTO.

Figure 25: TCP sequence/acknowledgement numbers versus time, observed at the ingress switch port of the labelled responder.

the initial RTO interval is equal to three times the initial RTT measurement.

The RTT measured during the connection establishment handshake suffers from two relevant problems. First, it does not provide any insight into the maximum possible queuing delay that the connection might experience. In an incast scenario, VOQ utilisation is volatile and can change dramatically within a single RTT on

account of the aggregate behaviour of responders.

Second, the small payload-less segments exchanged during the handshake can bias the measurement if serialisation delay forms a significant or dominant component of path RTT. Networks utilising short run copper or fiber optic cabling and low-latency network equipment can exhibit propagation delays that are shorter than the serialisation delay of full size frames. For example, at 10 Gbps a 78 B SYN frame serialises in 62 ns versus 1214 ns for a full size 1518 B TCP payload frame (more for jumbo frames which are used in some data centre and cluster networks). Commodity switches are capable of 350 ns port-to-port forwarding latency and fiber optic cabling is available with 4.80 ns/m propagation delay. Paths with serialisation delay as the dominant component of path latency can therefore exist today. Even with the more common port-to-port forwarding latency of a few microseconds found in larger scale switches, serialisation delay can contribute a non-trivial portion of the overall path latency.

When both problems conspire and an RTO interval that is inappropriately short is computed, the likelihood of spurious retransmits increase. This outcome is observed in Figure 25a with ARREAR enabled.

Responder 8's initial RTT measurement was 8.8 µs and derived initial RTO interval 26.3 µs, but it took 25.8 µs for its first packet of response data to arrive at the querier. The ACK elicited from the querier does not arrive at the responder until 2.7 µs after the spurious retransmit. Consider that if responder 8's initial RTT measurement had been based on a 1518 B segment, the calculated RTO interval would have been approximately 33 µs and the spurious retransmit avoided.

Fortunately, the time delta between retransmit and ACK was less than $\frac{RTT}{2}$ and the F-RTO logic was able to declare the restransmit as spurious and revert the TCP control block changes caused by the timeout. The nett effect is a minor increase in the transaction completion time by an amount equal to the serialisation delay of the retransmitted frame i.e. 1.2 µs.

Other scenarios examined were not so lucky, with the ACK arriving more than half an RTT after the retransmit. They retransmitted their entire initial window as a result and also spent the remainder of the transaction operating in congestion avoidance rather than slow start.

A more unfortunate outcome is observed in Figure 25b[14]. The queue that was

[14]Experiment utilised 8 responders returning a 162.3 kB cumulative response over a 10 Gbps network with 66 cells of VOQ buffering per port.

on the brink of saturation with ARREAR disabled was pushed over the edge by the single additional spuriously retransmitted packet. The second last packet of responder 8's response data was dropped and a RTO incurred as a result. Even though ARREAR ensured the RTO interval was appropriate for the scenario, the lack of a RTO with ARREAR disabled is why the ARREAR-enabled experiment completed later.

This investigation clearly indicates that SRTT/RTTVAR/RTO initialisation per RFC 6298 but without any lower bound can have undesirable effects in some incast scenarios. Further work is required to explore other potentially more appropriate RTO calculation methods for the data centre when high resolution RTT measurements are used (e.g., with ARREAR). Serialisation delay bias in measurements also warrants further investigation given the effects on TCP behaviour.

6.3.3 Possible Ways to Minimise Early Spurious Retransmits

Of the pairs with spurious retransmits examined in detail, a simple proof of concept patch that attached junk data to the SYN was enough to alleviate many initial window spurious retransmits. The receiver simply threw the data away, but it ensured serialisation delay was accounted for in the initial RTT measurement and therefore in the initial RTO interval.

Lowering the initial window from ten segments per RFC 6928 to three per RFC 3390 also alleviated the problem. By reducing the impulse load on the switch for a given number of responders, lowering the initial window allowed the ACKs for the initial window to return before the retransmit timer fired.

Although untested, packet pacing should also alleviate the problem and allow larger initial windows to be used for a given number of responders.

6.3.4 Measurement Resolution Versus RTT

As discussed in 6.1.1, TCP may have to contend with end-to-end path latencies ranging from seconds to picoseconds. By supporting measurements down to 233 ps resolution with the current implementation, ARREAR can utilise a wide range of clock sources.

Commodity data centre hardware typically has a selection of hardware clocks available, each with particular characteristics such as frequency, drift and read expense. The TSC found in most modern CPUs ticks at the CPU's base frequency,

which at commonly available speeds of 2–4 GHz provides nominal picosecond resolution. However, there are numerous complications associated with using the TSC in Symmetric Multi Processing (SMP) systems which are discussed in [267]. NICs can provide hardware timestamping features which negate the need for a high resolution general purpose system clock (e.g., 1.5 ns resolution [268]). An exploration of the clock resolution required for different plausible paths would provide some useful guidance for hardware specification or a hardware implementation of ARREAR.

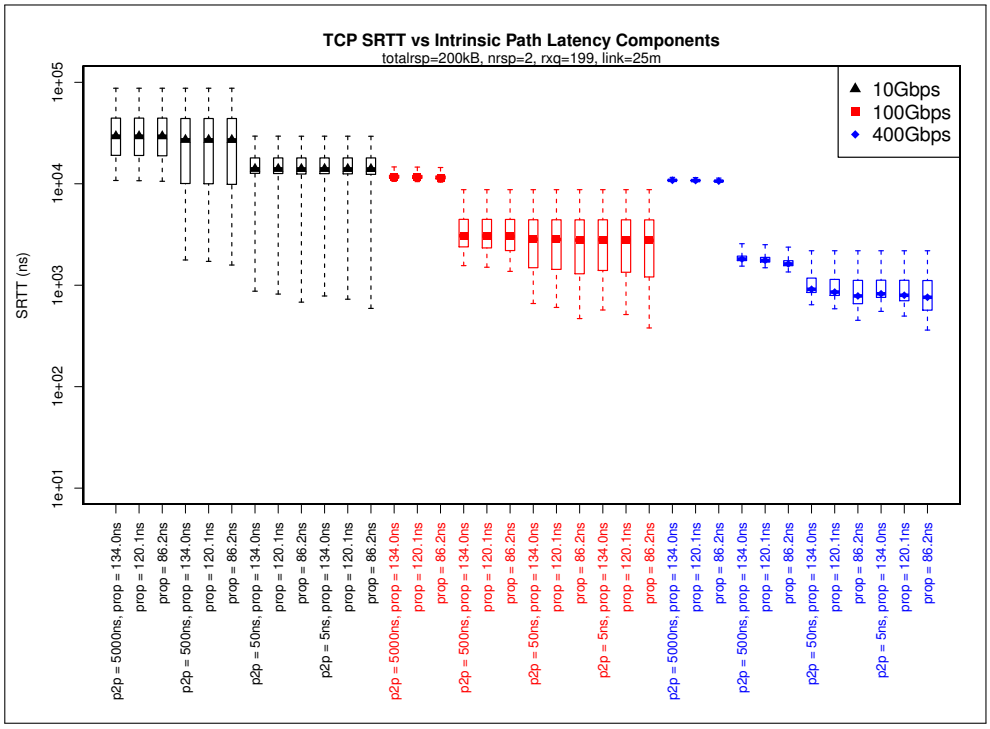


Figure 26: TCP SRTT versus path latency components over 25 m links.

Figures 26, 27 and 28 plot the range of TCP SRTT measurements made by ARREAR for a two responder, 100 kB cumulative response, 199 cells of VOQ buffering per port, lossless incast scenario. The focus is the intrinsic path latency components of signal propagation, serialisation and device forwarding so as to better comprehend their respective impact on transport layer RTT measurements.

Three distinct link distances are presented, with 25 m representing a room scale network, 5 m a rack scale network and 50 cm an intra-chassis scale network (distances are between node and switch i.e. half the end-to-end distance). Switch port-to-port forwarding latencies of 5 µs, 500 ns, 50 ns and 5 ns represent both current and plausible future network equipment latencies. Propagation delays associated with copper (5.36 ns/m), current fibre optic (4.80 ns/m) and future fibre optic (3.45 ns/m)

transmission media are represented. Link speeds of 10 Gbps, 100 Gbps and 400 Gbps similarly range between current and future possibilities.

Recall from 6.3.2 that the initial measurement made during the connection establishment handshake includes a minimal serialisation delay component on account of the small segment sizes exchanged. The larger the serialisation delay component of path latency is, the larger the difference between the initial RTT measurement and subsequent measurements made using MSS segments. This is clearly visible in Figures 26, 27 and 28 as the variable distance between the minimum (bottom tail) and 25th percentile box underside. As speed or port-to-port forwarding latency increase, so the relative serialisation delay component of path latency decreases and the minimum is closer to the bulk of the measurements.

Conversely, the maximum (top tail) represents the total measured path RTT inclusive of non-intrinsic latency components like queuing delay. Given that the response size and amount of VOQ buffering per port were held constant across all experiments, the relative differences between maxima provide insight into how the intrinsic components of path latency affect queue utilisation.

For algorithms that rely on precision of RTT measurements (e.g., delay-based

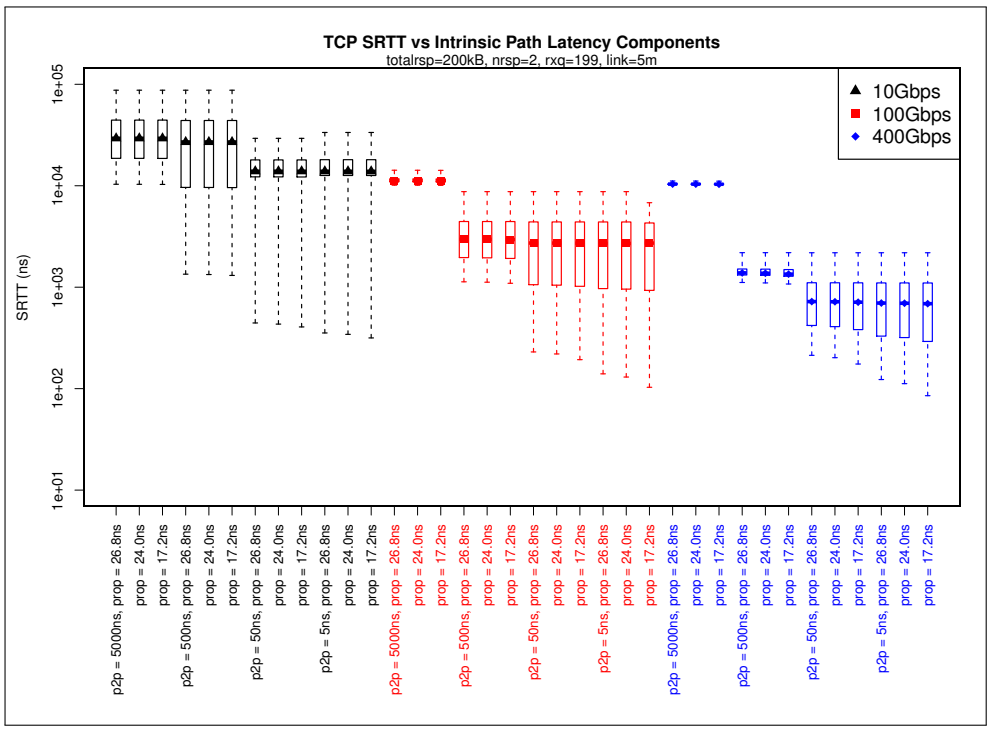


Figure 27: TCP SRTT versus path latency components over 5 m links.

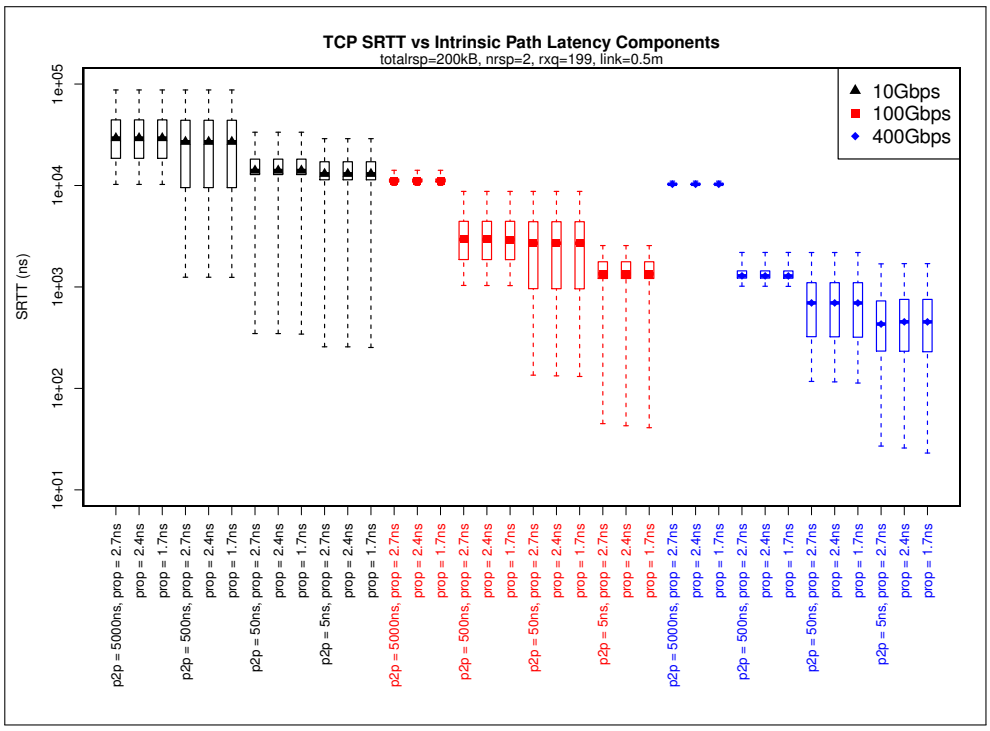


Figure 28: TCP SRTT versus path latency components over 50 cm links.

congestion control), having sufficient resolution to measure the full range is critically important. Given that the lower bound of the range determines the clock resolution requirement, the minimum path latencies observed are presented in Table 5 for the experiments that used the lowest switch port-to-port forwarding latency and transmission media latency. This subset of experiments provide a forward looking view of minimum path latencies, and suggest that double and eventually single digit nanosecond resolution should be sufficient for some time to come.

		Speed		
		10 Gbps	**100 Gbps**	**400 Gbps**
Distance	**25 m**	590.9 ns	379.0 ns	361.1 ns
	5 m	316.2 ns	103.1 ns	85.2 ns
	50 cm	253.1 ns	41.0 ns	23.1 ns

Table 5: Minimum path latencies measured with 5 ns switch port-to-port forwarding latency and 3.45 ns/m transmission media latency.

For incast RTO mitigation, measurement resolution over the complete path la-

tency range is not strictly required as discussed in 5.3.4. As long as TCP has sufficient resolution to see the tail end of queue build up leading to loss, it can still compute an appropriate RTO interval. However, the chosen clock must provide sufficient resolution to measure variance so that the RTO interval is not too short as also discussed in 5.3.4.

6.3.5 A Summary of ARREAR and its Experimental Evaluation

ARREAR is a sender-side, interoperable, and incrementally deployable RTT measurement scheme that equips TCP with the ability to measure path latencies based on the quality of a sender's local clock(s).

Conceptually, ARREAR treats the 32-bit TSval space as a pool of measurement tokens, with each increment of TSval to be made when a new RTT measurement is desired. Senders cache the TSval "token" value paired with a clock read, and when it returns as a TSecr, take a measurement by subtracting the cached clock read from the current time. The frequency with which measurements are taken and the TSval wrap period become elastic at the sender's discretion, which imparts useful flexibility. ARREAR also retains the TSval field semantics required for PAWS to be effective.

Without ARREAR, incast congestion induced RTOs are lower bounded to the stack's RTO_{min}, which is typically one or more orders of magnitude larger than the underlying path's RTT. Path-inappropriate RTO intervals lead to cumulative lost transmission opportunity with every RTO incurred, which can be further compounded in cases where binary exponential backoff is invoked. Throughput collapse occurs when responders experience RTOs concurrently, or when barrier-synchronisation requirements dictate any delayed responses must be waited on.

With ARREAR, incast congestion induced RTOs inflict less transmission opportunity loss on account of the path-appropriate RTO intervals in effect, and the improvement increases with increasing network speed and/or decreasing path latency. However, RFC 6298 RTO interval computation is not tuned for data centre speeds and latencies, and removing RTO_{min} was also shown to result in the computation of inappropriately short initial intervals which cause spurious retransmits.

The problems with initial RTO interval computation stem from two relevant problems associated with the RTT measured during the connection establishment handshake: it does not provide any insight into the maximum possible queuing delay that the connection might experience, and the small payload-less segments

exchanged during the handshake can bias the measurement if serialisation delay forms a significant or dominant component of path RTT. Attaching junk data to the SYN, lowering the initial window from ten segments per RFC 6928 to three per RFC 3390, and employing packet pacing should all be investigated to aid mitigation of this issue.

Finally, an investigation of clock resolution requirements for a range of plausible data centre path characteristics was conducted to provide guidance for hardware specification or a hardware implementation of ARREAR. Given that the lower bound of the RTT range determines the clock resolution requirement, the subset of experiments presented in Table 5 provide a forward looking view of minimum path latencies, and suggest that double and eventually single digit nanosecond resolution should be sufficient for some time to come.

7 Future Work

Pursuing the core contributions of this thesis has exposed a number of avenues for future work. CLUES opens up new opportunities to develop realistic traffic and hardware models, and would benefit from being able to leverage multithreading. RTO interval initialisation and computation needs to be revisited for high-bandwidth, low-latency paths. ARREAR opens up new opportunities to revisit past delay-based congestion control and related RTT-dependent control schemes, and actual implementations will require careful design for high performance.

Developing a more adequate multi-layered taxonomy and set of comparison criteria to fully categorise and compare the literature would be valuable, perhaps based on the efforts of Rojas-Cessa et al. [133].

CLUES would particularly benefit from the addition of validated data centre cross-traffic models, although with DCE [251] it could in theory run actual applications to generate real workloads. However, doing so would likely be more computationally expensive and therefore difficult to scale appropriately.

As a development environment, CLUES and the FreeBSD stack port would benefit from being able to alternate between single and multithreaded execution. This would allow the class of bugs currently excluded when running in CLUES to come into play at the developer's discretion. Having this capability would improve confidence in testing and simplify debugging by allowing concurrency to be easily checked for relevance to the bug.

The existing literature and discussion in Chapters 5 and 6 point to the long standing wisdom embodied in RFC 6298 needing to be revisited in the context of high-bandwidth, low-latency, relatively small BDP paths. What initial values should be used? If an RTO_{min} is defined, it would seem that it should take a variable rather than static form. How should it be calculated? Is the amount of hysteresis in SRTT and RTTVAR calculation appropriate? Is $SRTT + 4 \times RTTVAR$ an appropriate RTO interval calculation given that queue utilisation can be so volatile? Perhaps a larger amount of variance should be added. These issues and questions are yet to be thoroughly explored.

With low-latency paths and ARREAR-based RTT measurement resolution to match, delay-based congestion control and other schemes that rely on RTT as a control input should also be revisited. TIMELY [176] demonstrated convincing results by blending transport protocol congestion control with microsecond resolution

RTT measurements assisted by hardware offload capabilities. There are likely to be previously proposed schemes and entirely new schemes which could be reimagined in the cluster and data centre network context with access to appropriate resolution RTT measurements.

The issue of RTT measurement serialisation delay bias discussed in Chapter 6 warrants further research. The discussion clearly points to it being desirable for the first measurement made during the handshake to include maximum serialisation delay i.e. the SYN should carry a MSS worth of data. This is important to ensure an appropriate RTO interval during the initial window burst. The proof of concept patch developed that attached junk data to the SYN worked as expected, but was wasteful as it threw away the data at the receiver. TFO [63] could provide a standardised way to achieve this with actual payload data, but does require data to be ready to send at connect time.

There are also bias considerations for unidirectional versus bidirectional data transfer, as the TSecr returning in packets laden with data will take longer to return than pure ACKs, and could be subject to bandwidth asymmetry differences. Applications that tend to send variable sized segments (e.g., interactive shells like SSH) will also be more influenced by this bias. One could envisage a scheme that periodically probed serialisation delay using back-to-back segments of minimum and maximum size and compared the RTT measurement difference. Complications with delayed ACKs, hardware offload technologies and other issues would need to be considered.

The amount of sender-side state required by ARREAR could likely be reduced by exploring other related ideas. Logically splitting the TSval field into a measurement token sub-field and relative tick counter sub-field was one such idea briefly contemplated. This might allow a single base clock read to be periodically stored and used by multiple measurement tokens to calculate a delta.

An idea briefly contemplated to reduce ARREAR's computational overhead was to use the lowest order TSval bit as a flag to indicate the segment was a measurement candidate or not. This would avoid unnecessary map consultations in the fast path when measurements are not being taken for every segment. Another idea was to devise a TSC-based scheme for use as the clock source because of its relative cheapness to read. Many obstacles exist, but plausible work discussed with and undertaken by FreeBSD developer Matt Macy in the NextBSD project's "dctcp"

(data centre TCP) GitHub branch[15] has interesting potential to be integrated with ARREAR.

[15] https://github.com/NextBSD/NextBSD/tree/NextBSD/dctcp

8 Conclusion

The commoditisation and ubiquity of TCP/IP and related technologies has inevitably led to numerous growing pains from using them in environments they were not designed or optimised for. The use of TCP for high-speed, low-latency, large-scale cluster and data centre communications can trigger one such pain point – the incast congestion phenomenon and associated TCP throughput collapse pathology. Both have received a great deal of attention in line with the growing importance of the horizontally scaled TCP-based workloads that can induce them.

This thesis presented the CLUES toolkit, used it to perform an investigation of the incast phenomenon including related issues, and proposed the ARREAR scheme for TCP.

Chapter 4 presented CLUES, a hybrid simulation-emulation toolkit for cluster computing and data centre network research. CLUES combines the ns-3 discrete event network simulator, new validated models, NSC framework, and FreeBSD commodity open source operating system network stack. Many of ns-3's existing models (some with modifications) were leveraged in addition to the introduction of new models specific to cluster and data centre based experimental simulation.

A ns-3 QueryResponseApplication model was developed to provide a TCP-based, sequential, barrier-synchronised synthetic workload with which to trigger incast congestion. A ns-3 VOQ Ethernet switch model was created with relevant fundamental characteristics roughly comparable to those of commodity data centre switches. These models allowed parameters such as query/response size, transaction completion time, switch queue buffer size and switch scheduling to be investigated in detail.

A clean slate NSC network stack port based on FreeBSD-CURRENT was completed and integrated with CLUES. The port took in a significant amount of FreeBSD kernel infrastructure in an effort to maximise realism by minimising API emulation and modifications required to pristine sources. The stack's unmodified pristine base allows code to seamlessly transition between real test beds and CLUES, making it possible to maintain a single code base and more easily calibrate test environments.

The FreeBSD "VNET" virtualised network stack feature was leveraged to facilitate run-time stack instance allocation and sharing of common infrastructure between stack instances. At only 16 392 B of memory per instance, the CLUES stack supports ample scalability for experiments with large numbers of hosts.

The NSC API was modified to support measures that improve realism and stack scalability. Key changes included better targeted signals, fine-grained event scheduling, arbitrary route insertion and more flexible timekeeping.

The CLUES components were used to create the "incast" simulation, which serves as a practical instantiation of how to use CLUES to investigate a cluster and data centre specific issue. The simulation formed the experimental basis for the investigation of incast congestion in Chapter 5, and the evaluation of ARREAR in Chapter 6.

Chapter 5 detailed a CLUES-based exploration of incast congestion and related issues, with a focus on application-layer transaction completion time. Per-packet granularity data logging from the switch and end hosts provided detailed insights into the interactions between a number of variables and protocol behaviours. Variables of interest included Ethernet transmission speed, number of responders, response size, receive queue VOQ size and the RTT measurement resolution.

Retransmit timeouts, the cause of incast-induced TCP throughput collapse, were observed across a range of scenarios. The CLUES stack's RTO_{min} was removed, thereby allowing the RTO interval to fully adapt to the available RTT measurement resolution. This made it possible to fully observe the effects of RTO interval calculation for all experiment scenarios. Despite the modification, the 1 ms measurement floor imposed by TCP's RTT measurement machinery was shown to be insufficient for the simulated paths.

A set of 154729 experiments varying the number of responders, response size, Ethernet transmission speed and receive queue VOQ size were conducted. They provided a macro-level view of transaction completion times, showing distinct stratification of the data related to the number of non-parallel RTO events over a broad range of experiment permutations. The study showed incast-induced RTOs to be pervasive and their impact on transaction completion times increasingly undesirable as network speed increases.

It was shown that by improving the RTT measurement resolution, the RTO interval derived from the measured RTT could more appropriately adapt to the path. This improved transaction completion times, and therefore mitigated the effects of RTOs.

Chapter 6 proposed and evaluated an implementation of ARREAR, a sender-side, backwards compatible and incrementally deployable TCP RTT measurement scheme. ARREAR's development was guided by the fact that no practical incast

mitigation scheme is able to assert perfect knowledge about the end-to-end path. It is therefore important to adapt the end-to-end transport and its algorithms to the realities of best-effort IP networks operating at data centre speeds and latencies.

ARREAR equips TCP with the ability to measure path latencies based on the quality of a sender's local clock(s). Commodity hardware clocks are typically multiple orders of magnitude higher resolution than the artificially constrained 1 ms used by stacks to implement RFC 1323-based RTT measurement today. Accurate, path-appropriate resolution RTT measurements make it possible in turn to derive path-appropriate RTO intervals, and enables the use of RTT as a meaningful control input in non-core protocol mechanisms like delay-based congestion control.

Conceptually, ARREAR treats the 32-bit TSval space as a pool of measurement tokens, with each increment of TSval to be made when a new RTT measurement is desired. Senders cache the TSval "token" value paired with a clock read, and when it returns as a TSecr, take a measurement by subtracting the cached clock read from the current time. The frequency with which measurements are taken and the TSval wrap period become elastic at the sender's discretion, which imparts useful flexibility. ARREAR also retains the TSval field semantics required for PAWS to be effective.

ARREAR's co-opting of the TCP timestamp option to faciliate adaptive resolution RTT measurement meets all of the identified design goals. An implementation was developed against the CLUES stack and evaluated using the same "incast" simulation as Chapter 5.

A mirror data set to that from Chapter 5, but with ARREAR enabled, was created and compared. Of the experiment pairs that recorded different transaction completion times, the 10 Gbps experiments were split 27207 (shorter) to 45224 (longer), and 100 Gbps split 29920 (shorter) to 1328 (longer).

Shorter completion times stemmed from the high resolution RTT measurements translating into shorter, more appropriate RTO intervals. Experiments that experienced a RTO event without ARREAR therefore faired better with it, and the improvement increases with increasing network speed and/or decreasing path latency.

Longer completion times stemmed from spurious retransmits triggered by an inappropriately low RTO interval, especially during the initial window burst. As well as potentially exiting slow start prematurely, the spuriously retransmitted packets themselves induced loss in some scenarios. Further work is required to explore other

potentially more appropriate RTO calculation methods and the issue of serialisation delay bias in RTT measurements.

A final set of experiments were conducted to explore the effects of path delay components on TCP RTT measurement. The results provided insight into clock granularities required for plausible current and future networks with varying transmission speed, network equipment forwarding delay and propagation delay. The TCP measured SRTT distributions ranged below 100 ns minimums for some plausible future scenarios, suggesting that double and eventually single digit nanosecond resolution clocks should be sufficient for some time to come.

This thesis makes useful and concrete contributions towards improved data centre network research capability, and TCP's performance in the data centre. CLUES offers a sophisticated toolkit to inspire and support more detailed community research into data centre network protocol issues. ARREAR shows great potential to help reduce the negative application and user impact of RTOs on transaction completion times, which are commonly encountered with workloads that are prone to inducing incast congestion.

Acknowledgements

A great many people and organisations supported me and this work, helping to see it through to its very long awaited conclusion. I am deeply indebted to all of them, and immortalise below a wholly insufficient few words of gratitude.

Grenville Armitage

Thank you Grenville for being my coordinating supervisor and mentor. You took a chance on the young, clueless undergraduate that showed up at your office in 2003, and I am so grateful that you did. It has been a great privilege and honour to be embedded with you and CAIA in my many capacities. You have had a profound and formative impact on my life. Thank you for your tutelage, mentorship, encouragement, support, patience and friendship all these years. May good research and red wine continue to flow in equal measure for many more years to come.

Philip Branch

Thank you Philip for being my associate supervisor, and for the words of encouragement, insight and wisdom offered during our many thesis meetings.

David Fullagar and Netflix, Inc.

Thank you David for proffering the time off work and other material support that allowed me to finally get this thesis out of my (and everyone else's) life. Your extraordinary patience, understanding and support throughout this ordeal went far beyond what anyone could ever expect or hope for from their manager. I feel so incredibly privileged and lucky to work with you and the Netflix team.

Alan Ford, Warren Harrop and Randall Stewart

Thank you Alan, Warren and Randall for reviewing the draft thesis and providing valuable feedback. The final thesis is no doubt the better for it.

Matt Macy

Thank you Matt for your technical help and sounding board discussions over many years, on numerous topics and difficult bugs both directly and indirectly related to

this thesis.

Fred Baker and Cisco University Research Program Fund

This work was made possible in part by a gift from The Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation, for a project titled "Exploring possible mitigation for incast TCP congestion in data centres". Fred, particular thanks to you for your direct support of the URP project within Cisco Systems, Inc., and your continued support of CAIA and its mission over many years.

Greg Chesson, Sam Leffler, Kevin Yu and Google, Inc.

Thank you Greg for your support of my collaborative research visit to Google headquarters, and for your friendship during and after my time in Mountain View. The time spent with you working on ATC ultimately changed the course of my PhD and this thesis. Thank you Sam for connecting the dots and providing the introduction to Greg. Thank you Kevin for your help during the time you, Greg and I shared in our cozy little B43 office.

Jon Crowcroft, Andrew Moore and the University of Cambridge's Computer Laboratory

Thank you Jon for your support of my collaborative research visit to the Computer Laboratory and for our many fascinating discussions at the pub about anything and everything. Thank you Andrew for the material support to attend the MSN workshop at Cosener's House during my time in Cambridge, but a big thumbs down for not warning me about, or saving me from the supposedly mandatory "Sticky Carpets" experience.

Lincoln Dale and Arista Networks, Inc.

Thank you Lincoln for the talk you gave at CAIA on TCP incast and for providing your ns-3 modifications that provided inspiration and design guidance for the CLUES VOQ Ethernet switch model.

Dan Langille and BSDCan

Thank you Dan and BSDCan for the material support that allowed me to attend the 2009 and 2010 BSDCan conferences and co-located FreeBSD developer summits. My attendance allowed me to interact with FreeBSD developers and the community at large, as well as present some of my thesis related work. Both helped to cement my strong bonds with the FreeBSD community and improve my technical competency, which supported my completion of this work.

The FreeBSD Foundation

Thank you FreeBSD Foundation for contributing material support towards my 2010 BSDCan and FreeBSD developer summit attendance. Thank you also for indirectly supporting this work by funding a number of CAIA research and development projects that I was involved with.

Swinburne University of Technology and the Centre for Advanced Internet Architectures

I was deeply humbled and grateful to be awarded a Chancellor's Research Scholarship at the commencement of my PhD program. The scholarship provided material support for the first $3\frac{1}{2}$ years of my PhD and in particular, the amazing opportunity to undertake my collaborative research visits to the University of Cambridge's Computer Laboratory and Google headquarters in 2009.

The Centre for Advanced Internet Architectures (CAIA) was my base of operations and second home for more than a decade through undergraduate and then postgraduate life. Thank you past and present members and affiliates of CAIA for the camaraderie and helping to create a fantastic work environment.

Family and Friends

Finally, thank you, my dearest family and friends, for your love, encouragement and unwavering support through every high and low of this journey. We made it.

References

[1] R. Kohavi, R. M. Henne, and D. Sommerfield, "Practical guide to controlled experiments on the web: Listen to your customers not to the HiPPO," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 959–967. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281295

[2] J. Brutlag, "Speed matters for Google Web Search," Google, Inc., Tech. Rep., June 2009. [Online]. Available: http://services.google.com/fh/files/blogs/google_delayexp.pdf

[3] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, April 1965.

[4] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395–404, Jul. 1976. [Online]. Available: http://doi.acm.org/10.1145/360248.360253

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[6] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. New York, NY, USA: ACM, 2009, pp. 303–314.

[7] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2408776.2408794

[8] P. Baran, "On distributed communications: I. introduction to distributed communications networks," The RAND Corporation, Santa Monica, California, Tech. Rep. RM-3420-PR, August 1964.

[9] T. Marill and L. G. Roberts, "Toward a cooperative network of time-shared computers," in *AFIPS '66 (Fall): Proceedings of the November 7-10, 1966, fall joint computer conference*. New York, NY, USA: ACM, 1966, pp. 425–431.

[10] L. Roberts, "The evolution of packet switching," *Proceedings of the IEEE*, vol. 66, no. 11, pp. 1307 – 1313, nov. 1978.

[11] P. Baran, "Some perspectives on networks - past, present and future," in *Information Processing 77, Proceedings of the IFIP Congress 77*. North-Holland Publishing Company, August 1977, pp. 459–464.

[12] J. Postel, "Internet Protocol," RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[13] B. Carpenter, "Architectural Principles of the Internet," RFC 1958 (Informational), Internet Engineering Task Force, Jun. 1996, updated by RFC 3439. [Online]. Available: http://www.ietf.org/rfc/rfc1958.txt

[14] R. Bush and D. Meyer, "Some Internet Architectural Guidelines and Philosophy," RFC 3439 (Informational), Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3439.txt

[15] W. Willinger and J. Doyle, "Robustness and the internet: Design and evolution," March 2002.

[16] I. O. for Standardization, "Information technology - open systems interconnection - basic reference model: The basic model," International Organization for Standardization, International standard; ISO 9660 ISO 7498-1, November 1994.

[17] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[18] R. Braden, "Requirements for Internet Hosts - Application and Support," RFC 1123 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 2181, 5321, 5966, 7766. [Online]. Available: http://www.ietf.org/rfc/rfc1123.txt

[19] A. F. T. Committee, "ATM user-network interface (UNI) specification version 4.1," The ATM Forum, Specification af-atch-0193.000, 2002.

[20] CCITT, "Interface between dte and dce for terminals operating in the packet mode on public data networks," Comité Consultatif International Téléphonique et Télégraphique, Recommendation X.25, 1976.

[21] J. Postel, "User Datagram Protocol," RFC 768 (INTERNET STANDARD), Internet Engineering Task Force, Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[22] J. Postel, "Transmission Control Protocol," RFC 793 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[23] C. Hedrick, "Routing Information Protocol," RFC 1058 (Historic), Internet Engineering Task Force, Jun. 1988, updated by RFCs 1388, 1723. [Online]. Available: http://www.ietf.org/rfc/rfc1058.txt

[24] K. Lougheed and Y. Rekhter, "Border Gateway Protocol (BGP)," RFC 1163 (Historic), Internet Engineering Task Force, Jun. 1990, obsoleted by RFC 1267. [Online]. Available: http://www.ietf.org/rfc/rfc1163.txt

[25] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[26] J. Postel and J. Reynolds, "File Transfer Protocol," RFC 959 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1985, updated by RFCs 2228, 2640, 2773, 3659, 5797, 7151. [Online]. Available: http://www.ietf.org/rfc/rfc959.txt

[27] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945 (Informational), Internet Engineering Task Force, May 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1945.txt

[28] J. Postel and J. Reynolds, "Telnet Protocol Specification," RFC 854 (INTERNET STANDARD), Internet Engineering Task Force, May 1983, updated by RFC 5198. [Online]. Available: http://www.ietf.org/rfc/rfc854.txt

[29] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[30] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, 1984.

[31] G. Fairhurst, B. Trammell, and M. Kühlewind, "Services provided by IETF transport protocols and congestion control mechanisms," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-taps-transports-10, Mar. 2016, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-taps-transports

[32] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 756–769, 1997.

[33] C. Metz, "TCP over satellite... the final frontier," *Internet Computing, IEEE*, vol. 3, no. 1, pp. 76 –80, Jan/Feb 1999.

[34] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*. New York, NY, USA: ACM, 1988, pp. 314–329.

[35] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323 (Proposed Standard), Internet Engineering Task Force, May 1992, obsoleted by RFC 7323. [Online]. Available: http://www.ietf.org/rfc/rfc1323.txt

[36] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018 (Proposed Standard), Internet Engineering Task Force, Oct. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[37] A. Heffernan, "Protection of BGP Sessions via the TCP MD5 Signature Option," RFC 2385 (Proposed Standard), Internet Engineering Task Force, Aug. 1998, obsoleted by RFC 5925, updated by RFC 6691. [Online]. Available: http://www.ietf.org/rfc/rfc2385.txt

[38] M. Allman, "TCP Congestion Control with Appropriate Byte Counting (ABC)," RFC 3465 (Experimental), Internet Engineering Task Force, Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3465.txt

[39] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, updated by RFCs 5595, 5596, 6335, 6773. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[40] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," RFC 2960 (Proposed Standard), Internet Engineering Task Force, Oct. 2000, obsoleted by RFC 4960, updated by RFC 3309. [Online]. Available: http://www.ietf.org/rfc/rfc2960.txt

[41] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2002, pp. 89–102.

[42] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, 2006.

[43] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6824.txt

[44] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC 6182 (Informational), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6182.txt

[45] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal study of internet traffic in 1998-2003," in *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*. Trinity College Dublin, 2004, pp. 1–6.

[46] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *SIGCOMM '19: Proceedings of the ACM SIGCOMM 2010 conference on Data communication*. New York, NY, USA: ACM, 2010.

[47] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *Communications, IEEE Transactions on*, vol. 22, no. 5, pp. 637–648, May 1974.

[48] V. Cerf, Y. Dalal, and C. Sunshine, "Specification of Internet Transmission Control Program," RFC 675 (Historic), Internet Engineering Task Force, Dec. 1974, obsoleted by RFC 7805. [Online]. Available: http://www.ietf.org/rfc/rfc675.txt

[49] J. Postel, "Comments on Internet protocols and TCP," Aug. 1977. [Online]. Available: http://www.cis.ohio-state.edu/htbin/ien/ien2.html

[50] J. Postel, "DOD standard transmission control protocol," Dec. 1979. [Online]. Available: http://www.cis.ohio-state.edu/htbin/ien/ien124.html

[51] J. Postel, "DoD standard Transmission Control Protocol," RFC 761 (Historic), Internet Engineering Task Force, Jan. 1980, obsoleted by RFCs 793, 7805. [Online]. Available: http://www.ietf.org/rfc/rfc761.txt

[52] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896 (Historic), Internet Engineering Task Force, Jan. 1984, obsoleted by RFC 7805. [Online]. Available: http://www.ietf.org/rfc/rfc896.txt

[53] J. Nagle, "Congestion control in ip/tcp internetworks," *SIGCOMM Comput. Commun. Rev.*, vol. 14, no. 4, pp. 11–17, 1984.

[54] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, pp. 56–71, 1989.

[55] T. Berners-Lee, "Information management: A proposal," March 1989.

[56] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents," RFC 7414 (Informational), Internet Engineering Task Force, Feb. 2015, updated by RFC 7805. [Online]. Available: http://www.ietf.org/rfc/rfc7414.txt

[57] S. Floyd and M. Allman, "Specifying New Congestion Control Algorithms," RFC 5033 (Best Current Practice), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5033.txt

[58] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," RFC 5166 (Informational), Internet Engineering Task Force, Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5166.txt

[59] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681 (Draft Standard), Internet Engineering Task Force, Sep. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5681.txt

[60] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 3, pp. 304 –342, 2010.

[61] K. Mills, J. Filliben, D. Cho, E. Schwartz, and D. Genin, "Study of proposed internet congestion control algorithms," NIST, Tech. Rep., May 2010, nIST Special Publication 500-282. [Online]. Available: http://www.nist.gov/itl/antd/upload/NIST-SP-500-282.pdf

[62] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," RFC 6928 (Experimental), Internet Engineering Task Force, Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6928.txt

[63] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," RFC 7413 (Experimental), Internet Engineering Task Force, Dec. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7413.txt

[64] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540 (Proposed Standard), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7540.txt

[65] M. Belshe and R. Peon, "SPDY Protocol," Working Draft, IETF Secretariat, Internet-Draft draft-mbelshe-httpbis-spdy-00, Feb. 2012, work in Progress; replaced by draft-ietf-httpbis-http2. [Online]. Available: http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy

[66] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," Working Draft, IETF Secretariat, Internet-Draft draft-tsvwg-quic-protocol-02, Jan. 2016, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-tsvwg-quic-protocol

[67] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Internet Engineering Task Force, Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[68] T. Kelly, "Scalable tcp: improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, 2003.

[69] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-tcpm-cubic-01, Jan. 2016, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tcpm-cubic

[70] D. Leith, "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths," Working Draft, IETF Secretariat, Internet-Draft draft-leith-tcp-htcp-06, Apr. 2008, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-leith-tcp-htcp

[71] C. Jin, "FAST TCP for High-Speed Long-Distance Networks," Working Draft, IETF Secretariat, Internet-Draft draft-jin-wei-low-tcp-fast-01, Jul. 2003, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-jin-wei-low-tcp-fast

[72] M. Sridharan, K. Tan, D. Bansal, and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks," Working Draft, IETF Secretariat, Internet-Draft draft-sridharan-tcpm-ctcp-02, Nov. 2008, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-sridharan-tcpm-ctcp

[73] D. Clark, "Window and Acknowledgement Strategy in TCP," RFC 813 (Historic), Internet Engineering Task Force, Jul. 1982, obsoleted by RFC 7805. [Online]. Available: http://www.ietf.org/rfc/rfc813.txt

[74] B. Kernighan and D. Ritchie, *C Programming Language*. Pearson Education, 1988. [Online]. Available: https://books.google.com.au/books?id=Yi5FI5QcdmYC

[75] S. C. Johnson and D. M. Ritchie, "UNIX time-sharing system: Portability of C programs and the UNIX system," *Bell System Technical Journal*, vol. 57, no. 6, pp. 2021–2048, 1978. [Online]. Available: http://dx.doi.org/10.1002/j.1538-7305.1978.tb02141.x

[76] J. L. Yates, "Unix and the standardization of small computer systems," *BYTE Magazine*, vol. 08, no. 10, pp. 160–167, oct. 1983.

[77] R. Allan, *A History of the Personal Computer: The People and the Technology*. Allan Pub., 2001. [Online]. Available: https://books.google.com.au/books?id=FLabRYnGrOcC

[78] D. Hornby, B. Walker, and K. Pepple, *Consolidation in the Data Center: Simplifying IT Environments to Reduce Total Cost of Ownership.* Pearson Education, 2002.

[79] J. Dean, "The rise of cloud computing systems," in *SOSP History Day 2015*, ser. SOSP '15. New York, NY, USA: ACM, 2015, pp. 12:1–12:40. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2830903.2830913

[80] G. Lee, *Cloud Networking: Understanding Cloud-based Data Center Networks.* Elsevier Science, 2014. [Online]. Available: https://books.google.com.au/books?id=7QV0AwAAQBAJ

[81] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/945445.945450

[82] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[83] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 205–218. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298475

[84] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing.* Washington, DC, USA: IEEE Computer Society, 2004, p. 53.

[85] The FreeBSD Project, "The FreeBSD Project," Accessed 2 May 2016. [Online]. Available: http://www.freebsd.org/

[86] M. Zec, "Implementing a clonable network stack in the freebsd kernel," in *Proceedings of the 2003 USENIX Annual Technical Conference.* Berkeley, CA, USA: USENIX Association, 2003, pp. 137–150.

[87] P. Kamp and R. N. M. Watson, "Jails: Confining the omnipotent root," in *In Proc. 2nd Intl. SANE Conference*, 2000.

[88] Bjoern A. Zeeb, "vnet," Accessed 2 May 2016. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=vnet&manpath=FreeBSD+11-current

[89] The FreeBSD Project, "Contents of /head/sys/sys/linker_set.h," Accessed 2 May 2016. [Online]. Available: https://svnweb.freebsd.org/base/head/sys/sys/linker_set.h?view=markup

[90] R. Fujimoto, K. Perumalla, and G. Riley, *Network Simulation*, ser. Synthesis lectures on communication networks. Morgan & Claypool Publishers, 2007. [Online]. Available: https://books.google.com.au/books?id=YAFxmw6FHXAC

[91] K. Wehrle, M. Günes, and J. Gross, *Modeling and Tools for Network Simulation*. Springer Berlin Heidelberg, 2010. [Online]. Available: http://www.springer.com/in/book/9783642123306

[92] ns-2 Project, "The network simulator - ns-2," Accessed 2 May 2016. [Online]. Available: http://www.isi.edu/nsnam/ns/

[93] ns-3 Consortium, "ns-3," Accessed 2 May 2016. [Online]. Available: https://www.nsnam.org/

[94] OpenSim Ltd., "Omnet++ discrete event simulator," Accessed 2 May 2016. [Online]. Available: https://omnetpp.org/

[95] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, 2001.

[96] Simulcraft, Inc., "Omnest – high-performance simulation for all kinds of networks," Accessed 2 May 2016. [Online]. Available: https://omnest.com/

[97] Riverbed Technology, "Steelcentral riverbed modeler," Accessed 2 May 2016. [Online]. Available: http://www.riverbed.com/products/steelcentral/steelcentral-riverbed-modeler.html

[98] ns-3 Consortium, "What is ns-3," Accessed 2 May 2016. [Online]. Available: https://www.nsnam.org/overview/what-is-ns-3/

[99] ns-3 Consortium, "ns-2 & ns-3," Accessed 2 May 2016. [Online]. Available: https://www.nsnam.org/support/faq/ns2-ns3/

[100] ns-3 Consortium, "ns-3 documentation," Accessed 2 May 2016. [Online]. Available: https://www.nsnam.org/doxygen/

[101] SCons Foundation, "SCons: A software construction tool," Accessed 2 May 2016. [Online]. Available: http://www.scons.org/

[102] The OpenBSD Project, "The OpenBSD Project," Accessed 2 May 2016. [Online]. Available: http://www.openbsd.org/

[103] Free Software Foundation, Inc., "lwip – A Lightweight TCP/IP Stack," Accessed 2 May 2016. [Online]. Available: http://savannah.nongnu.org/projects/lwip/

[104] Linux Kernel Organization, Inc., "The Linux Kernel Archives," Accessed 2 May 2016. [Online]. Available: https://www.kernel.org/

[105] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.

[106] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-PDL-07-105, September 2007. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/Storage/CMU-PDL-07-105.pdf

[107] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*. New York, NY, USA: ACM, 2009, pp. 73–82.

[108] Y. Chen, R. Griffith, D. Zats, A. D. Joseph, and R. Katz, "Understanding TCP incast and its implications for big data workloads," *USENIX ;login: Magazine*, vol. 37, no. 3, pp. 24–38, 2012.

[109] G. Judd, "Attaining the promise and avoiding the pitfalls of TCP in the datacenter," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 145–157. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/judd

[110] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: ACM, 2009, pp. 65–72. [Online]. Available: http://doi.acm.org/10.1145/1592681.1592692

[111] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.

[112] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th annual conference on*

Internet measurement, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[113] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. New York, NY, USA: ACM, 2009, pp. 202–208.

[114] P. Devkota and A. L. N. Reddy, "Performance of quantized congestion notification in TCP incast scenarios of data centers," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug 2010, pp. 235–243.

[115] A. S. Anghel, R. Birke, D. Crisan, and M. Gusat, "Cross-layer flow and congestion control for datacenter networks," in *Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching*, ser. DC-CaVES '11. International Teletraffic Congress, 2011, pp. 44–62. [Online]. Available: http://dl.acm.org.ezproxy.lib.swin.edu.au/citation.cfm?id=2043535.2043542

[116] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 1377–1385.

[117] Z. Feng, B. Bai, B. Zhao, and J. Su, "An analytic goodput model for TCP incast," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, July 2012, pp. 427–432.

[118] H. Zheng and C. Qiao, "An effective approach to preventing TCP incast throughput collapse for data center networks," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec 2011, pp. 1–6.

[119] K. Chen, H. Zheng, Y. Zhao, and Y. Guo, "Improved solution to TCP incast problem in data center networks," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*, Oct 2012, pp. 427–434.

[120] G. Wang, Y. Ren, K. Dou, and J. Li, "The effect of the congestion control window size on the TCP incast and its implications," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, July 2013, pp. 000 625–000 628.

[121] M. Podlesny and C. Williamson, "An application-level solution for the TCP-incast problem in data center networks," in *Quality of Service (IWQoS), 2011 IEEE 19th International Workshop on*, June 2011, pp. 1–3.

[122] M. Podlesny and C. Williamson, "Solving the TCP-incast problem with application-level scheduling," in *2012 IEEE 20th International Symposium*

on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Aug 2012, pp. 99–106.

[123] D. Ke, R. Yongmao, and L. Jun, "Avoiding TCP incast through scheduling data requests," in *2012 Fourth International Conference on Multimedia Information Networking and Security*, Nov 2012, pp. 453–457.

[124] S. Kulkarni and P. Agrawal, *Analysis of TCP Performance in Data Center Networks*, ser. SpringerBriefs in Electrical and Computer Engineering. Springer-Verlag New York, 2014.

[125] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '98. New York, NY, USA: ACM, 1998, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/285237.285291

[126] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, "Comprehensive understanding of TCP incast problem," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1688–1696.

[127] 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 *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, Sep. 2008, pp. 1270 –1277.

[128] R. P. Tahiliani, M. P. Tahiliani, and K. C. Sekaran, "TCP variants for data center networks: A comparative study," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, Dec 2012, pp. 57–62.

[129] Y. Zhang and N. Ansari, "On architecture design, congestion notification, TCP incast and power consumption in data centers," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 39–64, First 2013.

[130] S. D. Pawar and P. V. Kulkarni, "A survey on congestion notification algorithm in data centers," *International Journal of Computer Applications*, vol. 108, no. 20, pp. 30–38, December 2014.

[131] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Network*, vol. 27, no. 4, pp. 22–26, July 2013.

[132] Y. Ren, Y. Zhao, P. Liu, K. Dou, and J. Li, "A survey on TCP incast in data center networks," *International Journal of Communication Systems*, vol. 27, no. 8, pp. 1160–1172, 2014. [Online]. Available: http://dx.doi.org/10.1002/dac.2402

[133] R. Rojas-Cessa, Y. Kaymak, and Z. Dong, "Schemes for fast transmission of flows in data center networks," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1391–1422, thirdquarter 2015.

[134] P. Sreekumari and J.-i. Jung, "Transport protocols for data center networks: a survey of issues, solutions and challenges," *Photonic Network Communications*, vol. 31, no. 1, pp. 112–128, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11107-015-0550-y

[135] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems," in *PDSW '07: Proceedings of the 2nd international workshop on Petascale data storage*. New York, NY, USA: ACM, 2007, pp. 1–4.

[136] S. Zhang, Y. Zhang, Y. Qin, Y. Han, Z. Zhao, and S. Ci, "OSDT: A scalable application-level scheduling scheme for TCP incast problem," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 325–331.

[137] Y. Yang, H. Abe, K. i. Baba, and S. Shimojo, "Staggered flows: An application layer's way to avoid incast problem," in *Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific*, Nov 2012, pp. 64–67.

[138] S. Osada, K. Kajita, Y. Fukushima, and T. Yokohira, "TCP incast avoidance based on connection serialization in data center networks," in *2013 19th Asia-Pacific Conference on Communications (APCC)*, Aug 2013, pp. 142–147.

[139] K. Kajita, S. Osada, Y. Fukushima, and T. Yokohira, "Improvement of a TCP incast avoidance method for data center networks," in *2013 International Conference on ICT Convergence (ICTC)*, Oct 2013, pp. 459–464.

[140] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 385–398. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482663

[141] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 1581–1589.

[142] S. Liu, W. Bai, H. Xu, K. Chen, and Z. Cai, "RepNet: Cutting tail latency in data center networks with flow replication," *CoRR*, vol. abs/1407.1239v2, 2015. [Online]. Available: http://arxiv.org/abs/1407.1239v2

[143] H. Zheng, C. Chen, and C. Qiao, "Understanding the impact of removing TCP binary exponential backoff in data centers," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, April 2011, pp. 174–177.

[144] D. Huo and Q. Cao, "Rto optimization for TCP incast scenarios," *International Journal of Advancements in Computing Technology*, vol. 3, no. 8, pp. 119–126, 09 2011.

[145] R. Miyayama, S. Osada, Y. Fukushima, and T. Yokohira, "Optimization of maximum timeout value in TCP with a fine-grained timer for data center networks," in *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2014, pp. 433–437.

[146] U. U. Hafeez, A. Kashaf, Q. u. a. Bajwa, A. Mushtaq, H. Zaidi, I. A. Qazi, and Z. A. Uzmi, "Mitigating datacenter incast congestion using RTO randomization," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[147] A. Kesselman and Y. Mansour, "Optimizing TCP retransmission timeout," in *Proceedings of the 4th International Conference on Networking - Volume Part II*, ser. ICN'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 133–140. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31957-3_17

[148] C. Deng, X. Wang, and S. Lu, "MIP: Minimizing the idle period of data transmission in data center networks," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 1179–1184.

[149] G. L. Chesson, "Aggregate Transport Control (ATC)," 2008, privately shared technical report.

[150] G. L. Chesson, "Aggregate transport control," U.S. Patent 8,339,957, 2012. [Online]. Available: http://www.google.com/patents/US8339957

[151] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in *Proceedings of the 6th International COnference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 13:1–13:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921186

[152] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, Apr. 2013. [Online]. Available: http://dx.doi.org.ezproxy.lib.swin.edu.au/10.1109/TNET.2012.2197411

[153] J. Hwang, J. Yoo, and N. Choi, "IA-TCP: A rate based incast-avoidance algorithm for TCP in data center networks," in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 1292–1296.

[154] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, "PAC: Taming TCP incast congestion using proactive ACK control," in *2014 IEEE 22nd International Conference on Network Protocols*, Oct 2014, pp. 385–396.

[155] L. Brakmo, "TCP-NV: Congestion avoidance for data centers," in *Linux Plumbers Conference 2010*, 2010. [Online]. Available: http://www.linuxplumbersconf.org/2010/ocw/sessions/525

[156] L. Brakmo, "TCP-NV: An update to TCP-Vegas," Google, Inc., Tech. Rep., August 2015. [Online]. Available: https://docs.google.com/document/d/1o-53jbO_xH-m9g2YCgjaf5bK8vePjWP6Mk0rYiRLK-U

[157] F. Zheng, Y. Huang, and D. Sun, "Designing a new tcp based on FAST TCP for datacenter," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 3209–3214.

[158] F. Zheng, Y. Huang, and K. Yin, "A delay-based TCP for solving incast problem in data centers," in *Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on*, July 2015, pp. 896–902.

[159] G. Wang, Y. Ren, K. Dou, and J. Li, "IDTCP: An effective approach to mitigating the TCP incast problem in data center networks," *Information Systems Frontiers*, vol. 16, no. 1, pp. 35–44, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10796-013-9463-4

[160] Y. Ren, J. Li, G. Wang, L. Li, and S. Shi, "SA-TCP: A novel approach to mitigate TCP incast in data center networks," in *2015 International Conference on Computing and Network Communications (CoCoNet)*, Dec 2015, pp. 420–426.

[161] J. Zhang and J. Wen, "RETCP: A ratio estimation approach for data center incast applications," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 338–343.

[162] S. Kulkarni and P. Agrawal, "A probabilistic approach to address TCP incast in data center networks," in *2011 31st International Conference on Distributed Computing Systems Workshops*, June 2011, pp. 26–33.

[163] S. Kulkarni, "Incast-free TCP for Data Center Networks," Ph.D. dissertation, Auburn University, 2012.

[164] A. S. W. Tam, K. Xi, Y. Xu, and H. J. Chao, "Preventing TCP incast throughput collapse at the initiation, continuation, and termination," in *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on*, June 2012, pp. 1–9.

[165] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses," Working Draft, IETF Secretariat, Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-01, Feb. 2013, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe

[166] Y. Cheng and N. Cardwell, "RACK: a time-based fast loss detection algorithm for TCP," Working Draft, IETF Secretariat, Internet-Draft draft-cheng-tcpm-rack-00, Oct. 2015, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-cheng-tcpm-rack

[167] C. Jiang, D. Li, M. Xu, and K. Zheng, "A coding-based approach to mitigate TCP incast in data center networks," in *2012 32nd International Conference on Distributed Computing Systems Workshops*, June 2012, pp. 29–34.

[168] 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*, vol. 32, no. 1, pp. 52–64, January 2014.

[169] S. K and K. V. K, "Mitigating incast congestion with LTTP for many to one communication in data centers," in *Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015 International Conference on*, March 2015, pp. 1–6.

[170] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348 (Proposed Standard), Internet Engineering Task Force, Sep. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5348.txt

[171] M. Luby, "LT codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

[172] L. Cheng, C. L. Wang, and F. C. M. Lau, "PVTCP: Towards practical and effective congestion control in virtualized datacenters," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[173] L. Cheng and F. C. M. Lau, "Revisiting TCP congestion control in a virtual cluster environment," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–1, 2015.

[174] P. Zhang, H. Wang, and S. Cheng, "Shrinking MTU to mitigate TCP incast throughput collapse in data center networks," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, April 2011, pp. 126–129.

[175] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency

in the data center," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 253–266. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/alizadeh

[176] R. Mittal, V. T. 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 *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 537–550. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2785956.2787510

[177] M. Miao, P. Cheng, F. Ren, and R. Shu, "Slowing little quickens more: Improving DCTCP for massive concurrent flows," in *Parallel Processing (ICPP), 2015 44th International Conference on*, Sept 2015, pp. 689–698.

[178] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can JUMP them!" in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/grosvenor

[179] M. Ghobadi and Y. Ganjali, "TCP pacing in data center networks," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, Aug 2013, pp. 25–32.

[180] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "SENIC: Scalable NIC for end-host rate limiting," in *Presented as part of the 11th USENIX Symposium on Networked Systems Design and Implementation*. Berkeley, CA: USENIX, 2014. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/senic-scalable-nic-end-host-rate-limiting-0

[181] Mellanox Technologies, "Mellanox CDNx Reference Architecture," 2016. [Online]. Available: http://www.mellanox.com/related-docs/solutions/cdn_ref_arch.pdf

[182] G. Parisis, T. Moncaster, A. Madhavapeddy, and J. Crowcroft, "Trevi: Watering down storage hotspots with cool fountain codes," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 22:1–22:7. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2535771.2535781

[183] J. Zhang, F. Ren, L. Tang, and C. Lin, "Taming TCP incast throughput collapse in data center networks," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[184] J. Hwang, J. Yoo, and N. Choi, "Deadline and incast aware TCP for cloud data center networks," *Computer Networks*, vol. 68, pp. 20 – 34, 2014, communications and Networking in the Cloud. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614000401

[185] A. M. Abdelmoniem and B. Bensaou, "Incast-aware switch-assisted TCP congestion control for data centers," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[186] Y. Lu and S. Zhu, "SDN-based TCP congestion control in data center networks," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, Dec 2015, pp. 1–7.

[187] J. Hwang, J. Yoo, S. H. Lee, and H. W. Jin, "Scalable congestion control protocol based on SDN in data center networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[188] A. Shpiner and I. Keslassy, "A switch-based approach to throughput collapse and starvation in data centers," in *Quality of Service (IWQoS), 2010 18th International Workshop on*, June 2010, pp. 1–9.

[189] R. Birke, D. Crisan, K. Barabash, A. Levin, C. DeCusatis, C. Minkenberg, and M. Gusat, "Partition/aggregate in commodity 10G ethernet software-defined networking," in *2012 IEEE 13th International Conference on High Performance Switching and Routing*, June 2012, pp. 7–14.

[190] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 25–36. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2413176.2413181

[191] J. Xu, H. Guo, J. Wu, J. Lin, D. Zhang, G. Chen, X. Zhang, and C. Chen, "SIG: Solution to TCP incast in SDN network based Openflow protocol," in *Asia Communications and Photonics Conference 2013*. Optical Society of America, 2013, p. AW4I.5. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=ACPC-2013-AW4I.5

[192] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM

'14. New York, NY, USA: ACM, 2014, pp. 503–514. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2619239.2626316

[193] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348 (Informational), Internet Engineering Task Force, Aug. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7348.txt

[194] P. Wang and H. Xu, "Expeditus: Distributed load balancing with global congestion information in data center networks," in *Proceedings of the 2014 CoNEXT on Student Workshop*, ser. CoNEXT Student Workshop '14. New York, NY, USA: ACM, 2014, pp. 1–3. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2680821.2680825

[195] H. Wang, Y. Xia, K. Bergman, T. E. Ng, S. Sahu, and K. Sripanidkulchai, "Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient *-cast connectivity," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 52–58, Jul. 2013. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2500098.2500105

[196] P. Samadi, V. Gupta, B. Birand, H. Wang, G. Zussman, and K. Bergman, "Accelerating incast and multicast traffic delivery for data-intensive applications using physical layer optics," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 373–374. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2619239.2631436

[197] C. Lee, Y. Nakagawa, K. Hyoudou, S. Kobayashi, O. Shiraki, and T. Shimizu, "Flow-aware congestion control to improve throughput under TCP incast in datacenter networks," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 3, July 2015, pp. 155–162.

[198] S. Jouet and D. P. Pezaros, "Measurement-based TCP parameter tuning in cloud data centers," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–3.

[199] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: Rapid precise packet loss notification in data centers," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 17–28. [Online]. Available: http://dl.acm.org.ezproxy.lib.swin.edu.au/citation.cfm?id=2616448.2616451

[200] J. Zhang, F. Ren, X. Yue, R. Shu, and C. Lin, "Sharing bandwidth by allocating switch buffer in data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 39–51, January 2014.

[201] A. Adesanmi and L. Mhamdi, "M21TCP: Overcoming TCP incast congestion in data centres," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, Oct 2015, pp. 20–25.

[202] "IEEE standards for local and metropolitan area networks: Supplements to carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - specification for 802.3 full duplex operation and physical layer specification for 100 mb/s operation on two pairs of category 3 or better balanced twisted pair cable (100BASE-T2)," *IEEE Std 802.3x-1997 and IEEE Std 802.3y-1997 (Supplement to ISO/IEC 8802-3: 1996; ANSI/IEEE Std 802.3, 1996 Edition)*, pp. i–324, 1997.

[203] "IEEE standard for local and metropolitan area networks–bridges and bridged networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec 2014.

[204] J. Jiang and R. Jain, "Analysis of backward congestion notification (BCN) for ethernet in datacenter applications," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007, pp. 2456–2460.

[205] J. Jiang, R. Jain, and C. So-In, "Forward explicit congestion notification (FECN) for datacenter ethernet networks," March 2007, IEEE 802.1au Meeting, Orlando, FL.

[206] C. So-In, R. Jain, and J. Jiang, "Enhanced forward explicit congestion notification (E-FECN) scheme for datacenter ethernet networks," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2008)*, Edinburgh, UK, June 2008, pp. 16–18.

[207] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, ser. HOTI '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 58–65. [Online]. Available: http://dx.doi.org/10.1109/HOTI.2010.26

[208] Y. Zhang and N. Ansari, "On mitigating TCP incast in data center networks," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 51–55.

[209] H. Shimonishi, J. Higuchi, T. Yoshikawa, and A. Iwata, "A congestion control algorithm for data center area communications," in *IEEE Communications Society Communications Quality and Reliability Workshop 2008*, Carefree, Arizona, USA, April 2008. [Online]. Available: http://www.ieee-cqr.org/2008/Day%202/Technical%20Session%201/A%20Congestion%20Control%20Algorithm%20for%20Data%20Center%20Area%20Communications.pdf

[210] V. S. Rajanna, S. Shah, A. Jahagirdar, C. Lemoine, and K. Gopalan, "XCo: Explicit coordination to prevent network fabric congestion in cloud computing cluster platforms," in *Proc. of 19th ACM International Symposium on High Performance Distributed Computing (HPDC), Chicago, Illinois*, 2010.

[211] V. S. Rajanna, S. Shah, and K. Gopalan, "XCo: Explicit coordination for preventing congestion in data center ethernet," in *Proc. of 6th IEEE International Workshop on Storage Network Architecture and Parallel I/Os*, 2010.

[212] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren, "Practical TDMA for datacenter ethernet," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 225–238. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168859

[213] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: Stability, convergence, and fairness," in *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '11. New York, NY, USA: ACM, 2011, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/1993744.1993753

[214] R. Stewart, M. Tüxen, and G. Neville-Neil, "An investigation into data center congestion control with ECN," BSDCan 2011, Tech. Rep., May 2011. [Online]. Available: http://www.bsdcan.org/2011/schedule/attachments/151_dc_cc.pdf

[215] T. Das and K. M. Sivalingam, "TCP improvements for data center networks," in *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2013, pp. 1–10.

[216] W. Chen, P. Cheng, F. Ren, R. Shu, and C. Lin, "Ease the queue oscillation: Analysis and enhancement of dctcp," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, July 2013, pp. 450–459.

[217] J. Hwang and J. Yoo, "FaST: Fine-grained and scalable TCP for cloud data center networks," *KSII Transactions on Internet and Information Systems*, vol. 8, pp. 762–777, 2014.

[218] Y. Xia, T. Wang, Z. Su, and M. Hamdi, "Preventing passive TCP timeouts in data center networks with packet drop notification," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 173–178.

[219] D. Zats, A. P. Iyer, G. Ananthanarayanan, R. Agarwal, R. Katz, I. Stoica, and A. Vahdat, "FastLane: Making short flows shorter with agile drop notification," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: ACM, 2015,

pp. 84–96. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2806777.2806852

[220] Y. Huang and B. Hu, "Enhanced DCTCP to explicitly inform of packet loss," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5511–5516.

[221] M. Kato, L. Eggert, A. Zimmermann, R. V. Meter, and H. Tokuda, "Extensions to FreeBSD datacenter TCP for incremental deployment support," BSDCan 2015, Tech. Rep., June 2015. [Online]. Available: https://www.bsdcan.org/2015/schedule/attachments/315_dctcp-bsdcan2015-paper.pdf

[222] M. Kato, "Improving transmission performance with one-sided datacenter TCP," Master's thesis, Keio University, 2013.

[223] P. Sreekumari, J.-I. Jung, and M. Lee, "An early congestion feedback and rate adjustment schemes for many-to-one communication in cloud-based data center networks," *Photonic Netw. Commun.*, vol. 31, no. 1, pp. 23–35, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1007/s11107-015-0526-y

[224] C. Liu and R. Jain, "Improving explicit congestion notification with the mark-front strategy," *Computer Networks*, vol. 35, no. 2-3, pp. 185 – 201, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128600001675

[225] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 50–61. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2018436.2018443

[226] N. Dukkipati, "Rate Control Protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Department of Electrical Engineering, Stanford University, 2007.

[227] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 115–126. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2342356.2342388

[228] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 127–138. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342389

[229] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 139–150, Aug. 2012. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2377677.2377711

[230] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "Deconstructing datacenter packet transport," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 133–138. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390254

[231] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486031

[232] L. Chen, S. Hu, K. Chen, H. Wu, and D. H. K. Tsang, "Towards minimal-delay deadline-driven data center TCP," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 21:1–21:7. [Online]. Available: http://doi.acm.org/10.1145/2535771.2535788

[233] 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 *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 2157–2165.

[234] A. Munir, I. A. Qazi, and S. B. Qaisar, "On achieving low latency in data centers," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 3721–3725.

[235] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 491–502. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2619239.2626305

[236] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and W. Sun, "PIAS: Practical information-agnostic flow scheduling for data center networks," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIII. New York, NY, USA: ACM, 2014, pp. 25:1–25:7. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2670518.2673871

[237] C. Ding and R. Rojas-Cessa, "DAQ: Deadline-aware queue scheme for scheduling service flows in data centers," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 2989–2994.

[238] L. Xu, K. Xu, Y. Jiang, F. Ren, and H. Wang, "Enhancing TCP incast congestion control over large-scale datacenter networks," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, June 2015, pp. 225–230.

[239] S. Fang, C. H. Foh, and K. M. M. Aung, "Prompt congestion reaction scheme for data center network using multiple congestion points," in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 2679–2683.

[240] J. Zhang, J. Wen, J. Wang, and W. Zhao, "TCP-FITDC: An adaptive approach to TCP incast avoidance for data center applications," in *Computing, Networking and Communications (ICNC), 2013 International Conference on*, Jan 2013, pp. 1048–1052.

[241] S. Shukla, S. Chan, A. S. W. Tam, A. Gupta, Y. Xu, and H. J. Chao, "TCP PLATO: Packet Labelling to Alleviate Time-out," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 65–76, January 2014.

[242] J. Huang, Y. Huang, J. Wang, and T. He, "Packet slicing for highly concurrent TCPs in data center networks with COTS switches," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, Nov 2015, pp. 22–31.

[243] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191 (Draft Standard), Internet Engineering Task Force, Nov. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1191.txt

[244] D. Guo, J. Xie, X. Zhou, X. Zhu, W. Wei, and X. Luo, "Exploiting efficient and scalable shuffle transfers in future data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 997–1009, April 2015.

[245] R. Scheffenegger, M. Kühlewind, and B. Trammell, "Additional negotiation in the TCP Timestamp Option field during the TCP handshake," Working Draft, IETF Secretariat, Internet-Draft draft-scheffenegger-tcpm-timestamp-negotiation-05, Oct. 2012, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-scheffenegger-tcpm-timestamp-negotiation

[246] R. Scheffenegger, M. Kühlewind, and B. Trammell, "Encoding of Time Intervals for the TCP Timestamp Option," Working Draft, IETF Secretariat, Internet-Draft draft-trammell-tcpm-timestamp-interval-01, Jul. 2013, work in Progress. [Online]. Available: http://tools.ietf.org/html/draft-trammell-tcpm-timestamp-interval

[247] D. Hayes, "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 100219A, 19 February 2010. [Online]. Available: http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf

[248] S. Jansen, "Network Simulation Cradle," Ph.D. dissertation, The University of Waikato, 2008.

[249] M. Lacage, "Experimentation tools for networking research," Ph.D. dissertation, Université de Nice-Sophia Antipolis, 2010.

[250] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, "Direct code execution: revisiting library OS architecture for reproducible network experiments," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 217–228.

[251] H. Tazaki, F. Urbani, and T. Turletti, "DCE cradle: Simulate network protocols with real stacks for better realism," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, ser. SimuTools '13. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 153–158. [Online]. Available: http://dl.acm.org.ezproxy.lib.swin.edu.au/citation.cfm?id=2512734.2512755

[252] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, Dec 2010, pp. 1–5.

[253] A. W. Malik, K. Bilal, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya, "CloudNetSim++: A toolkit for data center simulations in OMNET++," in *2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)*, Dec 2014, pp. 104–108.

[254] C. Minkenberg and G. Rodriguez, "Trace-driven co-simulation of high-performance computing systems using OMNeT++," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 65:1–65:8. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

[255] M. Zec and M. Mikuc, "Operating system support for integrated network emulation in IMUNES," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI*, 2004.

[256] Z. Puljiz and M. Mikuc, "IMUNES based distributed network emulator," in *2006 International Conference on Software in Telecommunications and Computer Networks*, Sept 2006, pp. 198–203.

[257] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the emulab network testbed," in *USENIX 2008 Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 113–128. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404023

[258] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[259] Z. Tan, Z. Qian, X. Chen, K. Asanovic, and D. Patterson, "DIABLO: A warehouse-scale computer network simulator using FPGAs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015, pp. 207–221. [Online]. Available: http://doi.acm.org.ezproxy.lib.swin.edu.au/10.1145/2694344.2694362

[260] G. Seguin, "Multi-core parallelism for ns-3 simulator," Tech. Rep., 2009.

[261] PANDUIT Corp., "Testing 10Gb/s Performance of Category 6 and 6A Structured Copper Cabling Systems within the Unified Physical Infrastructure^{SM} (UPI)," 2007. [Online]. Available: http://www.panduit.com/heiler/WhitePapers/WP-11%20UPI%2010Gb%20testing%20of%20copper%20HSDT%20structured%20cabling%20systems.pdf

[262] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6298.txt

[263] Arista Networks, Inc., "7150 Series 1/10GbE SFP Ultra Low Latency Switch Data Sheet." [Online]. Available: https://www.arista.com/assets/data/pdf/Datasheets/7150S_Datasheet.pdf

[264] F. Poletti, N. V. Wheeler, M. N. Petrovich, N. Baddela, E. Numkam Fokoua, J. R. Hayes, D. R. Gray, Z. Li, R. Slavík, and D. J. Richardson, "Towards high-capacity fibre-optic communications at the speed of light in vacuum." *Nature Photonics*, vol. 7, no. 4, pp. 279 – 284, 2013.

[265] T. Sato, K. Takeda, A. Shinya, M. Notomi, K. Hasebe, T. Kakitsuka, and S. Matsuo, "Photonic crystal lasers for chip-to-chip and on-chip optical interconnects," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 21, no. 6, pp. 728–737, Nov 2015.

[266] P. Andreades, Y. Wang, J. Shen, S. Liu, and P. M. Watts, "Experimental demonstration of 75 ns end-to-end latency in an optical top-of-rack switch," in *Optical Fiber Communications Conference and Exhibition (OFC), 2015*, March 2015, pp. 1–3.

[267] Gabriele Paolini, "How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures," 2010, Intel Corporation. [Online]. Available: http://www.intel.com.au/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[268] Chelsio Communications Inc., "Chelsio T5 Packet Rate Performance Report," 2014. [Online]. Available: http://www.chelsio.com/wp-content/uploads/2013/08/Chelsio-T5-Packet-Rate-Performance-Report.pdf

[269] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001, updated by RFCs 4301, 6040. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[270] J. Postel, "Internet Control Message Protocol," RFC 792 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 950, 4884, 6633, 6918. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt

[271] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335, 7053. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

Acronyms

ACK	ACKnowledgement
AIMD	Additive Increase Multiplicative Decrease
API	Application Programming Interface
AQM	Active Queue Management
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency NETwork
ARREAR	Adaptive Resolution RTT mEAsuRement
BDP	Bandwidth Delay Product
BSD	Berkeley Software Distribution
CC	Congestion Control
CLUES	CLUster nEtwork Simulation
CPU	Central Processing Unit
CSRG	Computer Systems Research Group
DSCP	Differentiated Services Code Point
ECMP	Equal-Cost Multi-Path
ECN	Explicit Congestion Notification
FCT	Flow Completion Time
FEC	Forward Error Correction
FIB	Forwarding Information Base
FIFO	First In First Out

FPGA	Field Programmable Gate Array
GPLv2	GNU General Public License v2
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IAB	Internet Architecture Board
IC	Integrated Circuit
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IEN	Internet Experiment Note
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRTF	Internet Research Task Force
ISA	Instruction Set Architecture
ISOC	Internet SOCiety
ITU	International Telecommunication Union
LBL	Lawrence Berkeley National Laboratory
LRO	Large Receive Offload
MAC	Media Access Control
MSL	Maximum Segment Lifetime
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit

NIC	Network Interface Card
NSC	Network Simulation Cradle
OS	Operating System
OWD	One Way Delay
PAWS	Protect Against Wrapped Sequence Numbers
PC	Personal Computer
PCP	Priority Code Point
PDU	Protocol Data Unit
PEP	Performance Enhancing Proxy
PFC	Priority Flow Control
PMTUD	Path MTU Discovery
QCN	Quantised Congestion Notification
QUIC	Quick UDP Internet Connections
RAM	Random Access Memory
RFC	Request For Comments
RIB	Routing Information Base
RTO	Retransmit Time Out
RTT	Round Trip Time
SACK	Selective ACKnowledgement
SCTP	Stream Control Transmission Protocol

SDN	Software Defined Networking
SIFTR	Statistical Information For TCP Research
SMP	Symmetric Multi Processing
SMSS	Sender Maximum Segment Size
SRTT	Smoothed Round Trip Time
TAPS	TrAnsPort Services
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TFO	TCP Fast Open
TFRC	TCP Friendly Rate Control
TSC	Time Stamp Counter
TSO	TCP Segmentation Offload
TTL	Time To Live
UDP	User Datagram Protocol
UMA	Universal Memory Allocator
VOQ	Virtual Output Queuing

Glossary

Additive Increase Multiplicative Decrease

A general control scheme for conservatively probing a limit and aggressively backing off if the limit is reached.

Application Programming Interface

A specification defining how computer code interacts with other code.

Active Queue Management

General term referring to management of network queues in a more intelligent manner than regular FIFO.

Address Resolution Protocol

Used to resolve the hardware MAC address associated with an IPv4 address of interest.

Advanced Research Projects Agency NETwork

A TCP/IP based packet switching network that was an early precursor to the Internet.

Adaptive Resolution RTT mEAsuRement

An adaptive resolution RTT measurement scheme for TCP.

Bandwidth Delay Product

A static measure of the maximum capacity of a path, in bits.

Berkeley Software Distribution

An open source operating system developed at UC Berkeley's CSRG.

CLUster nEtwork Simulation

A toolkit for cluster computing network simulation research.

Central Processing Unit

The general purpose IC that can perform programmed computation within a computer.

Computer Systems Research Group

A UC Berkeley research group known for developing the open source BSD operating system.

Differentiated Services Code Point

A 6-bit field in the IP packet header for assigning different classes of service to packets.

Equal-Cost Multi-Path

A packet forwarding strategy which uses multiple, equally good (in a packet forwarding metric sense) next hops to reach a given destination.

Explicit Congestion Notification

A signalling scheme that allows capable elements in the network to communication congestion information to endpoints, defined in [269].

Ethernet

Layer 2 variable length framing protocol developed at Xerox PARC.

Flow Completion Time

Length of time to complete a data exchange as measured at the network flow level.

Forward Error Correction

A network coding technique that transmits redundant data together with the intended data in a such a way that the intended data can be reconstructed in the face of some amount of corruption or loss.

Forwarding Information Base

A lookup table matching destination address to egress port.

First In First Out

A type of queue that writes to the back and reads from the front.

Field Programmable Gate Array

A reprogrammable IC capable of arbitrary computation..

FreeBSD

A UNIX-based open source operating system derived from UC Berkeley BSD.

GNU General Public License v2

A copy-left licence created for the GNU project.

Graphical User Interface

A computer interface allowing users to interact with a system visually using input hardware.

Hyper Text Transfer Protocol

An application-level protocol for distributed, collaborative, hypermedia information systems.

Internet Architecture Board

A committee of the IETF responsible for procedural and architectural oversight and guidance.

Integrated Circuit

An arrangement of transistors on a chip designed to perform some form of useful digital electronic computation.

Internet Control Message Protocol

An integral control protocol for IP networks defined in [270].

Institute of Electrical and Electronics Engineers

A professional organisation for engineers that provides many services and also develops data networking standards.

Internet Experiment Note

A series of technical documents modelled on the RFCs series that focused on early TCP/IP and Internet development matters.

Internet Engineering Task Force

An open membership collaborative organisation that develops data networking standards.

Internet Protocol

A protocol defined in [12] that specifies a best-effort addressing and forwarding scheme for use by packet-switched networks.

Internet Research Task Force

An open membership collaborative organisation that undertakes research in support of IETF activities.

Instruction Set Architecture

The set of instructions understood by a CPU.

Internet SOCiety

An open membership organisation that aims to promote the open development, evolution, and use of the Internet for the benefit of all people throughout the world..

International Telecommunication Union

A bureaucratic organisation that develops telecommunications and data networking standards.

Lawrence Berkeley National Laboratory

A University of California managed lab charged with conducting unclassified research across a wide range of scientific disciplines.

Large Receive Offload

An Ethernet hardware offload technology that allows the hardware to aggregate received segments and pass the chunk up the stack.

Media Access Control

A sub-layer of the data link layer related to addressing and channel access control.

Maximum Segment Lifetime

A concept first discussed in the early IETF standards documents to offer guidance to protocol implementors about the longest amount of time a segment might plausibly take to be delivered by an IP network.

Maximum Segment Size

The maximum amount of payload data a transport protocol segment can carry, which is dependent on the MTU and other lower layer protocols in use.

Maximum Transmission Unit

Largest size payload able to be put on the wire by the data link protocol.

Network Interface Card

A hardware addon card which connects a device to a network.

ns-2

Network Simulator v2.

ns-3

Network Simulator v3.

Network Simulation Cradle

Software framework for running external network stacks inside a network simulator.

Operating System

Base set of system software installed onto a computer responsible for managing hardware and providing common services to other software.

One Way Delay

Unidirectional delay between two endpoints.

Protect Against Wrapped Sequence Numbers

RFC 1323 defined mechanism for mitigating TCP stream corruption due to sequence number wrap.

Priority Code Point

A 3-bit field in the 802.1Q Ethernet header used to assign different priorities to frames.

Protocol Data Unit

Generic term for the encapsulation wire format associated with a protocol operating above the physical layer.

Performance Enhancing Proxy

A device that breaks and inserts itself in between the end-to-end connection (often transparently) in an attempt to improve some aspect of a connection.

Priority Flow Control

An 802.1Q Ethernet flow control mechanism.

Path MTU Discovery

A mechanism defined in [243] that allows end hosts to discover the minimum MTU of an arbitrary IP path to avoid sending packets that will require the network to fragment them.

POSIX

A family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.

Quantised Congestion Notification

An 802.1Q Ethernet congestion control mechanism.

Quick UDP Internet Connections

A UDP based congesstion controlled transport protocol being developed by Google.

Random Access Memory

Dynamic working memory used in computers for temporary storage.

Request For Comments

A document series containing technical and organizational notes about the Internet.

Routing Information Base

A lookup table matching destination address to next hop gateway address.

Retransmit Time Out

Length of time a TCP connection will wait for peer feedback before trying again.

Round Trip Time

A cumulative measure of the one way signal delay between two nodes in both directions.

Selective ACKnowledgement

A TCP loss recovery machanism specified in [36]..

Stream Control Transmission Protocol

An advanced feature set transport protocol defined in [271].

Software Defined Networking

A networking paradigm in which the forwarding and control plane are separated by a well defined, standardised interface.

Statistical Information For TCP Research

A FreeBSD kernel module providing event driven TCP connection information logging.

Symmetric Multi Processing

A computer architecture utilising multiple identical CPUs sharing the required system work.

Sender Maximum Segment Size

Maximum amount of payload data which can be sent in a TCP segment for a given connection.

SPDY

An application layer HTTP based protocol designed to overcome some pain points associated with delivery of web content over TCP.

Smoothed Round Trip Time

A weighted moving average over the last N RTT measurements, where N=8 for TCP.

sysctl

A syscall interface with which to set or get kernel settings, information and state addressed by way of a hierarchical tree structure named as a dotted set of components.

TrAnsPort Services

An IETF working group focused on improving the transport-layer abstractions for consumers.

Transmission Control Protocol

A connection-oriented, reliable, flow-controlled transport protocol defined in [22].

Time Division Multiple Access

A method for sharing a communications channel by allocating exclusive time slices to devices.

TCP Fast Open

An experimental proposal for TCP specified in [63] that allows data to be exchanged and consumed during the connection establishment handshake.

TCP Friendly Rate Control

A congestion control scheme specified in [170] for unicast flows operating in a best-effort environment that is reasonably fair when competing for bandwidth with TCP flows.

Time Stamp Counter

A monotonic clock found in many modern CPUs that ticks at the CPU frequency.

TCP Segmentation Offload

An Ethernet hardware offload technology that allows larger than MSS chunks of data to hit the hardware and be segmented on chip prior to transmission.

Time To Live

A concept commonly employed by network protocols to track data staleness e.g. the TTL field in the IP header is decremented at each hop and the packet discarded if the TTL reaches zero.

User Datagram Protocol

A connectionless datagram-based transport protocol defined in [21].

Universal Memory Allocator

The kernel slab allocator used in the FreeBSD kernel.

Virtual Output Queuing

A network device queuing scheme whereby the packets to be transmitted from a given output port are queued at a buffer associated with the input port the packet arrive on.

Index