Traffic re-engineering:

Extending resource pooling through the application of re-feedback

João Taveira Araújo

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Electronic & Electrical Engineering

University College London

2014

I, João Taveira Araújo, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Department of Electronic & Electrical Engineering

University College London

Abstract

Parallelism pervades the Internet, yet efficiently pooling this increasing path diversity has remained elusive. With no holistic solution for resource pooling, each layer of the Internet architecture attempts to balance traffic according to its own needs, potentially at the expense of others. From the edges, traffic is implicitly pooled over multiple paths by retrieving content from different sources. Within the network, traffic is explicitly balanced across multiple links through the use of traffic engineering. This work explores how the current architecture can be realigned to facilitate resource pooling at both network and transport layers, where tension between stakeholders is strongest.

The central theme of this thesis is that *traffic engineering* can be performed more efficiently, flexibly and robustly through the use of *re-feedback*. A cross-layer architecture is proposed for sharing the responsibility for resource pooling across both hosts and network. Building on this framework, two novel forms of traffic management are evaluated. Efficient pooling of traffic across paths is achieved through the development of an in-network congestion balancer, which can function in the absence of multipath transport. Network and transport mechanisms are then designed and implemented to facilitate path failover, greatly improving resilience without requiring receiver side cooperation. These contributions are framed by a longitudinal measurement study which provides evidence for many of the design choices taken. A methodology for scalably recovering flow metrics from passive traces is developed which in turn is systematically applied to over five years of interdomain traffic data. The resulting findings challenge traditional assumptions on the preponderance of congestion control on resource sharing, with over half of all traffic being constrained by limits other than network capacity.

All of the above represent concerted attempts to rethink and reassert traffic engineering in an Internet where competing solutions for resource pooling proliferate. By delegating responsibilities currently overloading the routing architecture towards hosts and re-engineering traffic management around the core strengths of the network, the proposed architectural changes allow the tussle surrounding resource pooling to be drawn out without compromising the scalability and evolvability of the Internet.

Acknowledgements

The work documented herein was made possible by my advisors along the way, formal or otherwise: Manuel Ricardo and Filipe Abrantes for inadvertently setting me on this path, George Pavlou and Miguel Rio for drafting directions which in my youth and to my chagrin I too often left unheeded, and Kensuke Fukuda for providing me with an eventful detour.

The ensuing comedy of errors was funded by the Portuguese government and would not have been complete without the following cast: Raúl Landa, Richard Clegg and Michio Honda as co-conspirators; Ioannis Psaras as proofreader; Suksant Sae Lor, Lorenzo Saino and Marinos Charalambides as fellow destitute Hackney residents; the entirety of NSRL for consistently getting me into the pub; Eleni Mykoniati for repeatedly getting us out; Pedro Cuba, Jorge Felizardo, Kalie Howse and Edgar Santos as comic relief. For support in absentia while my mind wandered: my parents, my family, Catarina Félix, Pedro Ribeiro, Nuno Salta.

To my mother for persevering in making me learn English,
otherwise writing this would have taken twice as long.
French on the other hand was of no use whatsoever.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Strategies for pooling traffic are locally applied by all stakeholders on the Internet in a bid to improve efficiency, resilience and flexibility. Operators resort to traffic engineering to load balance traffic across available network resources. Hosts adapt their sending rates to probe available network capacity. Peer-to-peer (P2P) applications often retrieve data chunks from multiple locations in order to efficiently distribute content amongst peers. Content providers can manipulate name resolution to balance demand across servers and hosting infrastructure. While these mechanisms share similar goals, they do so from different perspectives and as such may be at odds with each other.

This antagonism is played out within the Internet architecture as network, transport and application layers all attempt to influence how and where traffic flows. Against a backdrop of significant shifts in traffic patterns [CFEK06, CFEK08] and greater path diversity [TMSV03, BPS99, OZP+06], the issue of how best to balance traffic across multiple paths has become more relevant over time. Bolstered by strong theoretical groundwork [KV05, KMT07], support for enshrining traffic balancing at the transport layer has gained momentum, leading to recent efforts in the standardization of multipath transport [FRH+11]. The deployment of such protocols however is likely to be hindered by an operational reality; namely that most path diversity is within the network, and that most providers are unwilling to relinquish control of how traffic traverses their networks.

The network alone on the other hand appears incapable of managing traffic efficiently. For one, operators are constrained to balancing traffic transparently due to end host expectations. On the other hand, routers do not have enough knowledge of end-to-end traffic to make informed decisions on which path each packet should take. In many cases operators have enhanced their ability to manage traffic by extracting additional information per-packet, by looking beyond the network header, and per-flow, by reconstructing data streams over time, both of which ingrain protocol specific behaviour into the network. This increase of network awareness however comes at the expense of innovation at the edges, as developers become increasingly constrained in what type of protocols can be deployed.

Whether applied to providers wishing to reduce costs or hosts attempting to maximize throughput, the proliferation of unilateral solutions for resource pooling are manifestations of an underlying need. Rather than confine resource pooling to a single point of the Internet architecture and risk alienating a subset of stakeholders, this thesis explores how the existing Internet architecture can be extended to

accommodate both host and network requirements for resource pooling.

1.1 Problem statement

This thesis attempts to answer the following question:

> *Given the nature of Internet traffic, how can the current architecture be realigned to facilitate resource pooling at both network and transport layers?*

In proposing to *realign the current architecture*, the emphasis of any proposed solution must be applicable to the existing Internet architecture. The motivation for avoiding clean-slate solutions is largely due to the nature of the problem at hand. The different forms of resource pooling which are to be reconciled are as much a product of the Internet architecture as of its stakeholders. While it is clear that a clean-slate approach to resource pooling would have resulted in a different architecture, it may also have given rise to different stakeholders or different traffic patterns. By adhering to existing protocols, any potential solution can be directly applied and, by extension, validated.

While resource pooling is prevalent across all layers, the focus of this work is mostly restricted to reconciling *network* and *transport* layers. Most forms of resource pooling above the network layer will attempt to benefit the end user, while below the transport layer most resources within a single administrative domain will conspire towards the same ends. It is at the intersection of network and transport layers where the juxtaposition of interests is greatest within the Internet architecture.

Facilitating resource pooling however should not dictate an outcome in the tussle between network and hosts, but rather provide an architecture within which such a tussle can evolve. In some cases balancing traffic solely from the hosts may be desirable, while in other cases providers may wish to retain full control. Both represent extremes of a range of outcomes which should be possible within a unifying architecture.

Finally, designing an efficient resource pooling architecture must take into account the *nature of Internet traffic*. While scaling Internet traffic poses considerable technical challenges, understanding its emergent properties plays a pivotal role in simplifying traffic management. Any solution presented must not only address future traffic needs but also exploit its properties.

1.2 Contributions

This thesis contains the following contributions:

A mutualistic architecture which identifies re-feedback as a potential solution for bridging different forms of resource pooling. Path Re-Feedback and Loss Exposure (PREFLEX) is unique in exposing network path diversity to hosts, while making the network aware of performance metrics which are instrumental to effective traffic engineering.

A model for balancing congestion is derived which enables providers to make more efficient use of available end-to-end capacity. Compared to existing multipath transport protocols, balancing congestion from within the network can support legacy applications as well as flows which are too

short to effectively explore path diversity. The proposed model is evaluated and shown to outperform traditional traffic engineering methods without the need for per-flow state within the network.

A novel methodology for flow reconstruction is detailed which recovers transport behaviour from passive traffic traces. This includes a mechanism for round trip time (RTT) recovery based on cumulative histogram construction and peak detection and a scalable process for systematically classifying flow throughput dynamics.

A longitudinal analysis of Internet traffic spanning approximately 5.7 billion flows collected over five years. Compared to previous studies, this work documents macroscopic shifts in the nature of Internet traffic – *where* traffic originates from – as well as performing a significant reappraisal of commonly held assumptions on *how* the Transmission Control Protocol (TCP) behaves in practice.

A resilient traffic management framework which can be unilaterally deployed by edge domains. The proposed solution, INFLEX, is modelled upon software-defined networking principles while drawing on the insights afforded by all aforementioned contributions above. The resulting system is shown to provide fast, scalable, end-to-end fault detection with low overhead at either the end-host or the network. Finally, extensions are described which allow INFLEX to equally perform congestion balancing – resulting in a complete architecture for dynamic traffic management at the edge requiring only sender-side modifications.

1.3 Publications

Software-defined network support for transport resilience

J. Taveira Araújo, R. Landa, R. G. Clegg and G. Pavlou

IEEE/IFIP Network Operations and Management Symposium (NOMS) 2014

A longitudinal analysis of Internet rate limitations

J. Taveira Araújo, R. Landa, R. G. Clegg, K. Fukuda and G. Pavlou

33rd IEEE International Conference on Computer Communications (INFOCOM) 2014

MALAWI: Aggregated longitudinal analysis of the MAWI dataset

J. Taveira Araújo, K. Fukuda

7th ACM CoNEXT Student Workshop 2011

Balancing by PREFLEX: Congestion Aware Traffic Engineering

J. Taveira Araújo, I. Grandi, R. G. Clegg, M. Rio and G. Pavlou

10th International IFIP TC 6 Networking Conference (NETWORKING) 2011

A mutualistic resource pooling architecture

J. Taveira Araújo, M. Rio and G. Pavlou

3rd ACM International Workshop on Re-Architecting the Internet (ReArch) 2010

The Need for Congestion Exposure in the Internet

T. Moncaster, L. Krug, M. Menth, J. Araújo, S. Blake, R. Woundy

draft-moncaster-conex-problem-00, IETF Internet draft 2010

1.4 Thesis Outline

This thesis is organized as follows:

Chapter 2 provides an overview of how resource pooling has evolved across different layers of the Internet architecture, detailing how network resources are pooled end-to-end, through congestion management, and how traffic is balanced across multiple paths.

Chapter 3 proposes PREFLEX, a resource pooling architecture which accommodates both congestion control and traffic engineering through the application of re-feedback.

Chapter 4 builds upon the foundation of the previous chapter and models and evaluates a functional congestion balancer, allowing the network to perform dynamic, adaptive traffic engineering by crowd sourcing information from end hosts.

Chapter 5 presents a longitudinal study on Internet interdomain traffic and characterizes the impact of structural changes in content distribution on underlying traffic patterns.

Chapter 6 follows on from the previous chapter, delving into the nature and evolution of rate limitations affecting traffic exchanged over TCP.

Chapter 7 revisits the architectural concepts introduced in chapters 3 and 4 in light of the findings presented in chapters 5 and 6. A unilaterally deployable solution providing scalable, resilient traffic management is presented modelled upon recent advances in software-defined networking.

Chapter 8 draws conclusions on the present work and posits potential directions for future work.

Chapter 2

Resource pooling

The Internet was designed around the ability to pool shared resources. From an initial desire to share mainframes, resource pooling has grown to encompass statistical multiplexing within links and across paths, and is indigenous to every layer of the network stack [WHB08]. This chapter briefly reviews how resource pooling has evolved within the Internet architecture and proposed future directions. Section 2.1 reviews how capacity is shared end-to-end, and how ensuing congestion is managed while section 2.2 focuses on how traffic is allocated and balanced across different network paths.

2.1 Pooling end-to-end: congestion management

Much of the value of the Internet is derived from the ability to pool capacity end-to-end: the statistical multiplexing afforded by packet switching resulted in a more efficient network, far better suited to the bursty nature of computer traffic than prevalent circuit switched alternatives. Where demand outstrips supply however, congestion arises. This is the inevitable consequence of sharing scarce resources, and as such should not be viewed as a problem in and of itself. The difficulty lies in resolving this contention both efficiently and fairly.

2.1.1 Historical precursors

Early work in defining congestion, and forever ingraining the term in networking nomenclature, can be traced to work on queuing systems by Kleinrock [Kle75] beginning from 1961. However, it is in the work of Paul Baran at RAND [Bar64b] that a deeper concern for the implications of congestion can be found.

Baran's work on a "distributed network concept" [1] was intended for military purposes, and as such was required to deal with traffic overload gracefully. Traffic was expected to be marked by users according to a military precedence system. During times of traffic overload, Baran envisioned a priority control console, operated directly by a military officer, which would regulate the allocation of capacity to each traffic class. Control was in the network. In part, this reflected military chain of command, but Baran had uncovered an additional cause for concern which would later resurface in the Internet: during times of crisis, users could not be trusted to mark their own traffic:

[1] The term packet switching was yet to be adopted. It would later be imported from similar work by Donald Davies and others at National Physical Laboratory, UK (NPL).

Messages once labelled "Deferred" will be stamped "Operational Immediate," and jammed back into the input hopper. It is analogous to the inflationary competition of competing buyers for scarce goods. We temporarily delude ourselves into thinking that we are buying more capability by inflating the precedence indicator.

Long before such notions would become widespread, Baran had both identified an economic context to congestion and the inherent flaws in precedence marking which would later malign the Internet Protocol (IP). Underlined in the document are two further passages:

Communications networks are rarely exercised in real-time to simulate extensive communications network damage and overload.

A communication network that does not let the user know how long it will be before his message will be delivered to the end addressee may be theoretically oscillatory.

Both would prove prescient as the Advanced Research Projects Agency Network (ARPANET) took off in 1969. The ARPANET was at the helm of an ideological battle between packet switching and circuit switching, and much early work would revolve around the search for an architectural identity. While commonly identified as the precursor to the Internet, the ARPANET was akin to connection oriented networks such as X.25. Messages would be relayed between Interface Message Processors (IMPs) towards hosts, with each IMP ensuring reliable, in-order message delivery.

In the midst of debates over layering or the relative merits of connection and connectionless architectures, there was little interest in understanding how network resources should be shared. By and large, there was also no need. A combination of abundant capacity, slow terminals and inefficient data protocols ensured that contention was a rare occurrence. Initially, hosts could only send one outstanding message at a time, resulting in poor performance which would decrease proportionally to the number of hops in a connection. By 1971, the peak burst rate recorded between hosts was 40kb/sec using parallel connections, approximately 50% of bottleneck capacity [Cer74]. Even within such systems, loss, however rare, could prove problematic, and its detection was being debated by Postel and others in 1973 [Pos73, Hat73]. It was becoming evident that existing flow control was inefficient. Additionally, end-to-end acknowledgements would be necessary, as Baran had predicted almost a decade in advance. The death knell for the connection-oriented architecture would come from France, where the CYCLADES [Pou73] network had demonstrated the feasibility of building reliable communications on top of unreliable network elements. The simplicity and elegance of this *datagram* network and associated windowed flow control were quickly adopted within ARPANET.

The specification of the TCP [CI05] in 1974 pushed connection establishment, sequencing, flow control and message reassembly towards the end-host. The architectural blueprint for what would become known as the Internet was almost complete, but not without some loose ends. For one, the *Internetwork header* conflated seemingly different functions. By 1975, efforts had begun to separate the IP header from the TCP header [2]. Likewise, the nature of congestion was still not thoroughly understood.

[2] Vestigial traces of this late separation are still evident - to this day the calculation of the TCP checksum takes into account the IP header.

When referring to TCP retransmissions, Cerf and Kahn point out that "the HOST level retransmission mechanism (...) will not be called upon very often in practice. Evidence already exists that individual networks can be effectively constructed without this feature". In an assessment of ARPANET protocols from the same year [Cer74], Cerf lists under unresolved problems and issues that "the IMP subnet must have a way of combating congestion which may result from too rapid influx of data". This mechanism would later materialize as the Internet Control Message Protocol (ICMP) Source Quench option [Pos81], whereby routers could notify senders to reduce their rates. Alongside TCP flow control, it would prove to be the only means of managing congestion as Network Control Program (NCP) was finally replaced by TCP/IP on Flag Day, 1982.

That more attention was not paid to managing congestion in the eight years between the initial specification of TCP and its eventual deployment is unfortunate. However, there was little indication as to the potential gravity of the problem. By 1982, there were still only 235 hosts on the Internet [Lot92]. Most of these would have been attached to packet switches over protocols providing flow control such as X.25. Furthermore, there were more pressing concerns elsewhere. Work was under way in refining applications which would drive initial demand. Addressing and routing posed scalability concerns as it became clear that the number of potential networks would rapidly exceed the limit of 256 seen as sufficient only a few years earlier [CI05]. The host table was becoming too large to distribute as a single file and so distributed alternatives were needed [Moc87, BLNS81]. With no progress on any of these it is unlikely the Internet would have become large enough to become a victim of its own success.

Finally, extensively testing how the network would operate under load was never an option. Unlike contemporary packet switched networks such as CYCLADES, the ARPANET had always been too *useful* to be anything but a production network [Day10]. Within four years of Flag Day the number of hosts would increase tenfold and connectionless Local Area Networks (LANs) would become widespread. By 1986, congestion collapse was recurrent as hosts persisted in retransmitting packets into an overloaded core. Baran's vision of a packet-switched network had come full circle.

2.1.2 TCP congestion control

The importance of congestion control within TCP was first highlighted by Nagle [Nag84] in 1984. Working at Ford Motor Company, which operated the only private TCP/IP long-haul network at the time, Nagle reported periods of excessive load and unusual congestion problems. Nagle proposed alterations to address the small-packet problem, in which an excessive number of small packets were generated for the transmission of keyboard strokes, and to improve the handling of ICMP Source Quench messages. Neither would prove sufficient in the long run. As issues arising from congestion increased, so too did the interest within the community to address the problem. Independent work by Jain and Ramakrishnan at Digital Equipment Corporation (DEC), Phil Karn, Lixia Zhang and Craig Partridge had all begun to converge on similar solutions for managing congestion, but ultimately it would be Van Jacobson who would synthesize the model for congestion control which is still prevalent today.

In [Jac88], Van Jacobson begins by asserting that for a system to be stable each connection should abide by a "packet conservation principle": a new packet should not enter the system until an old packet

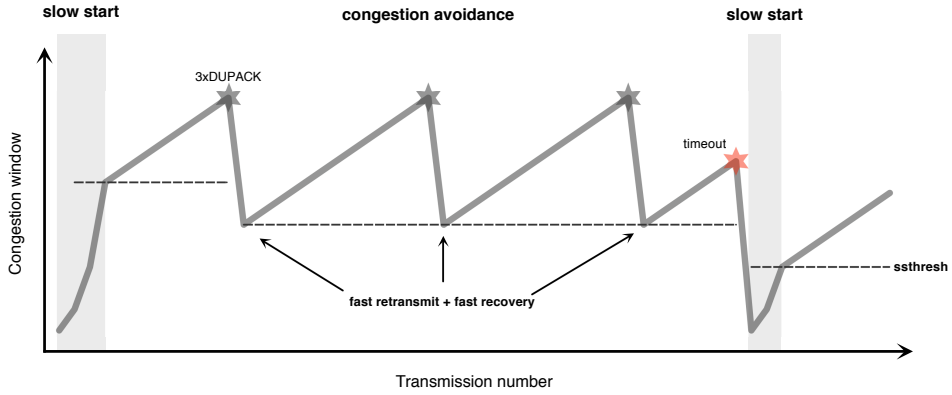


Figure 2.1: TCP congestion control.

leaves [3]. Building on this intuition, two separate moments in a connection lifetime are identified, as shown in figure 2.1.

Initially, a connection must attempt to reach equilibrium by ramping up its window size during a *slow-start* phase. From an initial *congestion window* (cwnd), a TCP connection increases the congestion window by one packet for each acknowledgement (ACK) received, until congestion is detected or the window size surpasses the *slow-start threshold* (ssthresh), an initially predefined system variable. The resulting increase in the congestion window is exponential, and thus expected to exceed available capacity at most by a factor of 2. Prior to the introduction of *slow-start*, this overshoot was potentially much higher as hosts would initiate a connection by immediately adjusting to the receiver's advertised window.

Upon exiting *slow-start*, a flow is deemed close to equilibrium. The TCP sender moves to the *congestion avoidance* phase, where an Additive Increase Multiplicative decrease (AIMD) strategy is used to probe for available bandwidth. In the absence of congestion, the window is increased by one Maximum Segment Size (MSS) every round trip time. If congestion is detected, the window is halved and slow-start threshold (ssthresh) is readjusted to this value.

To signal congestion Van Jacobson settled on packet loss. While loss could arise from data corruption, such cases were considered sufficiently rare as not to hinder efficiency. Loss provided a poor resolution signal, but was the only form of congestive feedback shared by all networks [4]. The efficiency of retransmission was further improved by providing a more accurate estimate of the RTT, and including a mechanism for *fast retransmit*. The resulting changes were quickly deployed as TCP Tahoe and proved instrumental in curbing congestion.

Congestion control would go on to become a staple of networking research. Successive refinements, including the addition of *fast recovery* resulted in TCP Reno [Ste97] and then TCP NewReno [PAS99], which would become the most commonly accepted and widely deployed congestion control algorithm for over a decade. Variants such as Vegas [BOP94] and Fast AQM Scalable TCP (FAST) [WJLH06]

[3] While often attributed to Van Jacobson, the concept of packet conservation was not new. Its origins can be traced at least as far back as 1972, when Donald Davies proposed a similar "isarithmic" approach for controlling congestion within networks [Dav72].

[4] ICMP source quench, despite being widely deployed, was not a viable solution for reasons described in section 2.1.4.

would complement the use of loss with the estimation of queuing delay, and adjust rates accordingly. Binary Increase Congestion Control (BIC) and CUBIC [XHR04, HRX08] addressed rate adjustment as a binary search problem and would be adopted by Linux distributions. Finally, Compound TCP (CTCP) [TS06], deployed from Windows Vista onwards, would use loss and delay as a combined congestion signal, tracking both as separate components before calculating an overall congestion window.

In spite of these changes, the core of TCP congestion control algorithms can still be directly traced to Van Jacobson's initial proposal. That a stop-gap solution has managed to sustain the growth of the Internet for over two decades is an outcome even Van Jacobson could not have anticipated. In [Jac88], concern is expressed in that "the congestion control noise sensitivity is quadratic in w but it will take at least another generation of network evolution to reach window sizes where this will be significant". Despite work to address such inherent scaling issues [Mat09, Kel03], TCP has remained unchanged - as bandwidth increases so too will the time between loss events. Likewise, while TCP shares bandwidth efficiently, there was no claim the outcome would be fair. In describing future work, Van Jacobson observes that "only in gateways, at the convergence of flows, is there enough information to control sharing and fair allocation". Over the coming years, congestion management would slowly seep into the network.

2.1.3 Traffic shaping

The introduction of congestion control in TCP addressed the growing pains as the Internet scaled to become a network of networks. Congestion however plays an equally important role within networks. On a short time scale, there is a variable cost which an operator must pay its own providers for transit traffic. On a longer time scale, there is a fixed cost in upgrading capacity at each provisioning cycle. By minimizing congestion, operators hope to reduce both.

From a provider's perspective, congestion is commonly defined as the average utilization of a link over some period of time. Under this definition, the most congested link does not necessarily experience the highest packet loss, but rather exhibits the highest utilization over an extended period of time. This reflects the volume-based pricing models typically used between networks, the most common of which is 95th percentile pricing. Under such a scheme, aggregate traffic rates are calculated over each time window of duration t. At the end of each billing cycle, time windows are ordered by traffic rate, of which the highest 5% are discarded. The largest remaining traffic rate represents the 95th percentile rate that is used to charge customers. Inbound and outbound traffic are usually tracked separately, of which only the most significant is charged.

Both hosts and network remain intent in reducing their own view of congestion, but in doing so pursue potentially disparate objective functions. Over time, the changing nature of Internet traffic, technology and stakeholders has exacerbated this tussle [CWSB05]. In [BCL09], Bauer et al. characterize the evolution of Internet traffic into three phases:

- Prior to the 1990s, the Internet was largely confined to academic and research purposes. Congestion typically occurred within the core, as long haul lines would often trail behind local area networks in terms of bandwidth. Traffic was dominated by a small number of applications - mail,

bulk file transfer and low bandwidth interactive sessions - all of which were tolerant of congestion. Furthermore, the Internet was still small enough that congestion was viewed as a shared concern to be jointly addressed by the community.

- By the 1990s the commercialization of the Internet was in full swing, attaining mass market status as residential users signed up for dial-up access. The dramatic increase in user base was somewhat attenuated by the existence of a bottleneck at the edge. Dial-up offered low data rates of up to 56.6kb/s and intermittent connectivity, both of which were instrumental in curbing demand. The most significant new application protocol, Hyper Text Transfer Protocol (HTTP), put more demands on end systems than on the network, as web servers scaled to cope with thousands of simultaneous requests. Congestion within the network was rare and most likely to occur at data modem banks which would in effect perform admission control - excess subscribers attempting to connect would receive a busy signal.

- Since 2000, the advent of broadband access lifted bandwidth constraints at the edge and pushed congestion deeper into Internet Service Providers (ISPs). Higher data rates coupled with "always on" connectivity led to a much larger set of applications placing different demands on the network, from bandwidth hungry file sharing to congestion sensitive Voice-over-IP (VoIP) or video streaming. Network provisioning, previously driven by the needs of commercial clients, would become increasingly driven by the usage patterns of residential customers.

The proliferation of media-rich applications led to a growing disparity between customers. A minority of "heavy" users incurred the majority of costs from pooling available bandwidth despite paying the same flat fee as remaining "light" users. In the long term, capacity upgrades would prove ineffective in quenching demand from the former group while passing unwarranted costs onto the latter. Faced with a deluge of data and having to satisfy the needs of a diverse range of both applications and customers, network operators inevitably took it upon themselves to manage congestion.

The Internet architecture however does not provide the means for a network to manage congestion effectively. Furthermore, the communal approach to addressing problems had since grown into a cumbersome and convoluted standardization process within the Internet Engineering Task Force (IETF) [Bus05]. Unsurprisingly, operators quickly resorted to piecewise solutions to artificially stifle demand with little consideration on the wider impact of their actions. These attempts at *traffic shaping* would take on a variety of forms.

One method of reducing congestion is by limiting the total volume of traffic a subscriber can send or receive over a given period and applying penalties should this cap be exceeded. This method imposes the least technical requirements as volume accounting is already widely performed. While volume capping introduces an expected upper bound for the volume of traffic traversing a network over each billing cycle, it fails to address demand patterns over shorter time scales. Subscribers within their volume allowance can still cause considerable congestion by acting synchronously, for example by being active during peak hours only. Conversely, volume capping unnecessarily penalizes applications which shift their traffic towards less congested periods.

Alternatively, congestion can be potentially reduced by throttling throughput. By imposing rate limits during periods of oversubscription, providers may attenuate the impact any single traffic source can have on others. Doing so efficiently however is challenging. For one, available capacity and periods of congestion cannot easily be narrowed to a single throughput threshold or time period. In provisioning rates for the worst expected case, providers forsake efficiency when contention may not arise. Furthermore, throttling can be performed on a per-customer or per-flow basis. In the former case, throttling fails to take into account how much shared capacity a customer has occupied over time, degrading performance equally for heavy and light users alike. Throttling flows on the other hand may prioritize types of traffic for applications using specific port ranges. Applications affected by such network policing will however tend to adapt over time, either by changing port or increasing the number of flows to bypass throttling.

Volume capping and rate throttling are necessarily naive and inefficient as they can function on network layer information alone, but in doing so are largely benign. Either method however can be performed selectively by inspecting packet contents in order to prioritize traffic. These attempts at Deep Packet Inspection (DPI) have had a much more nefarious effect on the Internet. The introduction of DPI middleboxes break layering, introduce complexity within the network and hinder the deployment of new, innovative applications and transport protocols at the edge. Applications targeted by DPI have become increasingly stealthy, often masquerading as HTTP traffic. The ensuing arms race between traffic obfuscation and traffic identification methods has been waged at great cost to the Internet as a whole and little benefit to any stakeholder in particular.

In addressing the tussle surrounding congestion, transport and network communities applied successive independent fixes rather than reaching a compromise within a common architectural solution. Over time, the damage caused in adopting the former has only strengthened the arguments for the latter.

2.1.4 Explicit congestion control

In previous sections unilateral solutions to congestion management were reviewed, where either end-hosts or networks perform their duties in isolation. The overbearing dominance of the transport layer in resource sharing was shown to be an ex post construct: neither Baran [Bar64a], Cerf [CI05] or, for that matter, Van Jacobson [Jac88] envisaged an Internet where control was exerted from the edges alone. This section reviews proposals where both endpoints and network share responsibility for managing congestion through the use of explicit congestion control, whereby sources adjust sending rates according to explicit network feedback.

There have been enough failed attempts at deploying explicit congestion control on the Internet that it has become common to overlook the fact it was once the dominant model for congestion management. Prior to TCP congestion control, hosts performed end-to-end flow control alone. While most underlying data layer protocols provided some degree of flow control, gateways could signal overload by sending ICMP source quench messages to sources. Hosts were then expected to reduce sending rates upon receiving this explicit feedback from the network. Unfortunately, the source quench mechanism was marred with oversights. The specification of ICMP[Pos81] stated that source quench messages “may”

be sent by gateways for every packet it discards. However it also allowed source quench messages to be emitted by gateways approaching, rather than exceeding, their capacity limit. As a result, upon being cautioned by a gateway, a sender had no indication of whether the packet at fault had been discarded. Under such circumstances, the source host "should cut back the rate (...) until it no longer receives source quench messages from the gateway", but no further indication on how such a goal can be attained was provided. Confusingly, destination hosts could also emit source quench messages if overcome by processing, duplicating functionality already present at the transport layer [Pos80].

Source quench was inevitably deprecated [Bak95], crippled by overloaded semantics and a lack of obligation or guidance for sender compliance. In retrospect, source quench messages represented an attempt at doing *too little, too early* [CHM+03]. Endpoint congestion control would only come of age with TCP Tahoe. Prior to the continuous control theory mechanisms introduced by Van Jacobson, Jain, and others, there was no framework within which explicit feedback could be modelled. Only with the onset of congestion collapse events did many of the frailties of source quench messages become apparent. There was no understanding of the problem at hand until it was too late.

Despite this failed first attempt, interest in applying explicit network feedback to the Internet did not wane. In 1988 Jain and Ramakrishnan proposed an explicit binary feedback scheme for congestion avoidance which would evolve into DECbit [RJ90]. Under such a scheme, the network header would contain a single bit which could be toggled by intermediate routers experiencing the onset of congestion. The resulting bit would then be returned to the sender. If at least half of the previous congestion window had been marked, the sender would decrease its window by 7/8, otherwise it would continue to increase its sending rate additively.

DECbit proved promising enough to be ported to IP with minor changes as the Explicit Congestion Notification (ECN) protocol [Flo94]. ECN would reuse two bits from the Type of Service (ToS) field to retrieve binary network feedback. If neither bit was set, a packet is deemed to be non-ECN Capable Transport (ECT) and therefore legacy traffic. If either bit was set, both hosts had established the usage of ECN during the initial handshake and the packet belongs to an ECT flow. Having detected the onset of congestion according to a given Active Queue Management (AQM) discipline, an ECN aware queue would then set both bits to encode the Congestion Experienced (CE) code point. Unlike DECbit, a single CE marking would be sufficient motive for a sender to decrease its congestion window, making ECN compatible with probabilistic AQM mechanisms such as Random Early Detection (RED) [FJ93]. Furthermore, the multiplicative decrease was more conservative than in DECbit, and followed the existing TCP standard of halving the congestion window. This conscious decision to make ECN enabled traffic perform similarly to legacy TCP over large time scales would strip it of a potentially important competitive advantage in deployment whilst retaining one of the less scalable components of TCP.

Both DECbit and ECN proved to be *too little, too late*. Proposed at the same time as TCP congestion control, DECbit was initially shunned in favour of a practical solution which could be rapidly deployed and rectified. By the time sufficient experience with binary explicit feedback had been gained through deployment in DECnet, the Internet had dramatically changed. The standardization of ECN [RFB01]

arrived in an Internet overrun by commercial interests which neither understood the problem nor valued the solution. As the Internet scaled in size, so too did its inertia to change, ossifying architecturally around a core set of protocols. Each new addition would be required to address an otherwise unassailable problem. In an era of dial-up access and rapid roll-out of improvements in TCP congestion control, ECN seemed like a costly anachronism. Deployment was further hampered by broken TCP implementations and routers which could adversely affect connection establishment for ECN enabled traffic. Despite improvements [Kuz05] ECN remains poorly used and turned off by default on hosts and routers alike [BBB11].

Ironically for a protocol which has been plagued by deployment issues, the widespread availability of ECN in commodity network equipment may yet reassert its relevance. Proposals such as re-ECN [BJCG$^+$05] leverage ECN in order to achieve cost fairness. Variable-structure congestion Control Protocol (VCP) [XSSK08] overrides ECN semantics to provide an additional bit of congestion information per packet. Data Center TCP (DCTCP) [AGM$^+$10] repurposes ECN to achieve high throughput with low buffer usage within a data centre environment. Rather than reacting to each marking as a potential loss event, DCTCP sources reduce their window proportionally to the *fraction* of marked packets. From a sequence of single bits, DCTCP derives and acts upon a multi-bit signal. Both DCTCP and VCP retrofit multi-bit network feedback, itself the subject of much research.

The first concerted attempt at deploying multi-bit network feedback would arise in Asynchronous Transfer Mode (ATM) networks. ATM provided a connection-oriented data link layer over which service categories with different Quality of Service (QoS) constraints would coexist. In addition to addressing precise throughput and delay requirements through the Constant Bit Rate (CBR) and Variable Bit Rate (VBR) services, ATM offered an Available Bit Rate (ABR) service [BF95], catering for applications with vague requirements and intended as a cost-effective means of supporting data traffic. Applications using ABR would specify ranges of acceptable rates which the network would then satisfy through the dynamic allocation of available bandwidth amongst virtual circuits.

ATM was primarily driven by telecom operators who embraced the concept of explicit network control of rates. Capitalizing on similar work on DECbit, an Explicit Forward Congestion Indication (EFCI) code point was integrated into the ATM data header to provide binary feedback on congestion along a path. In parallel however, research progressed on rate-based feedback, where the network would explicitly provide sources with the allowed sending rate. Since ATM used small, fixed size packets, called *cells*, control information was preferentially provided through different payload types. As a result, ABR sources in addition to sending standard data cells would periodically emit a Resource Management (RM) cell to communicate requirements and retrieve information from intermediate network nodes. The RM cell contained fields for different parameters of interest, such as the Minimum Cell Rate (MCR), Peak Cell Rate (PCR), Additive increase to rate (AIR) or Rate decrease factor (RDF). Upon receiving a RM cell, ATM switches would readjust the cell payload to reflect local conditions if they could not meet the specified requirements. The information contained within the updated RM cell would then be fed back from the destination to the source, which would readjust their rates accordingly.

ABR would prove a prolific source of research in explicit network feedback. As ATM waned in popularity however, so too did research efforts within the context of ABR. In its wake, a wide set of proposals for traffic management, including credit-based and rate-based approaches, had been investigated, often concurrently [Jai96]. With little consensus and no unifying vision however, the ABR rate-based framework would encompass different mechanisms with often overlapping functionalities: *too much, too early*.

Multi-bit network feedback would resurface applied to Internet traffic with the eXplicit Congestion Protocol (XCP) [KHR02]. Taking a clean slate approach, Katabi et al. revisited congestion control without the constraints imposed by existing network equipment. The resulting proposal introduces a shim *congestion header* between the IP and TCP headers carrying estimates of the flow RTT and inter-packet time [KF07]. Additionally, a *delta_throughput* field contains the allowed increase in throughput, and is set to the maximum value at the source. As a packet traverses routers, the value of *delta_throughput* is updated by routers according to local conditions. The feedback loop is then closed as destinations copy the value of the inbound *delta_throughput* field into the outbound *reverse_feedback* field. Upon receiving network feedback, the source updates its congestion window according to the function:

$$cwnd = max\left(cwnd + reverse\_feedback \cdot RTT, MSS\right) \tag{2.1}$$

While functionally similar to the rate-based framework proposed in ABR, XCP represented a significant step forward in explicit congestion control. For one, XCP bypassed the need for flow state which was assumed in ATM networks. Secondly, XCP specified a cohesive mechanism for redistributing bandwidth as opposed to expecting a variety of methods to coexist. An Efficiency Controller (EC) periodically calculates the amount of bandwidth F to be distributed over the following control interval d according to:

$$F = \alpha \cdot (C - input\_bw) - \beta \cdot \frac{q}{d} \tag{2.2}$$

where C is the link capacity, $input\_bw$ is the inbound throughput and q is the persistent queue length. Constants α and β are set to ensure stability by weighing the amount of spare bandwidth allocated and the rate at which the queue is drained. The available bandwidth F is then distributed amongst flows by the Fairness Controller (FC), which calculates the $delta\_throughput$ to be applied for each packet:

$$delta\_throughput = \begin{cases} \frac{F}{N} \cdot \frac{X}{d} & \text{if } F > 0 \\ F \cdot \frac{s}{input\_bw \cdot d} & \text{if } F \leq 0 \end{cases} \tag{2.3}$$

where N is the approximate number of flows, calculated as the sum of inter packet time X normalized by control interval d. If the resulting $delta\_throughput$ is smaller than the existing value in the congestion header, the router updates the field accordingly before forwarding the packet.

Using multi-bit feedback XCP achieves high efficiency and stability with negligible queuing delay. In prioritizing queue drainage and system stability however XCP presented a natural performance bias towards long flows. Under high utilization, only a small proportion of bandwidth would be reshuffled resulting in a potentially slow convergence to flow fair rates for short-lived flows. Recognizing the

importance such flows had in overall user experience, Dukkipati et al. proposed the Rate Control Protocol (RCP). Building on the explicit congestion control framework proposed by XCP, RCP emulates processor sharing by emphasizing the Average Flow Completion Time (AFCT) rather than delay stability [DKZSM05]. Rather than calculating a change to the source congestion window (cwnd), the equivalent FC in an RCP system calculates the explicit maximum rate R to be set for flows traversing a router at interval n:

$$R_n = R_{n-1} + \frac{F}{N} \tag{2.4}$$

where N, the approximate number of active flows, is obtained by:

$$N = \frac{C}{R_{n-1}} \tag{2.5}$$

By setting a single rate for all flows within an RTT of flow start, RCP rapidly redistributes bandwidth, but does so at the expense of queue stability. RCP proved well suited for web traffic where completion time is often of greater importance than jitter. As with XCP, the requirement for both end systems and bottleneck routers to be overhauled would prove too high a barrier for incremental deployment. To blame the lack of deployment of either solution on hardware requirements alone however would be disingenuous, and ignoring the most valuable lesson of all. Both offered a compelling universal solution to a problem now escalating with pluralism: *too much, too late*. Many of the shortcomings in TCP congestion control are inherent to limitations in the amount of information one can infer from loss. In hindsight, that proposals as disparate as XCP and RCP had emerged in spite of a wealth of explicit network feedback signalled a growing lack of consensus within the community on how to move beyond Van Jacobson's legacy. By 2007, a tongue-in-cheek rebuttal of flow fairness [Bri07] would only widen this rift. For a growing minority, the absence of an overarching solution to managing congestion was no longer construed as an inability to provide the right answer, but as a symptom of asking the wrong question.

2.1.5 Congestion exposure

Resource sharing has evolved haphazardly to being performed by end-hosts and network alike. This apparent duplication of effort and source of potential conflict suggests a rare lack of foresight within the Internet architecture. More confounding however is that unlike other architectural mishaps, such as the conflation of locater and identifier within addressing, the absence of progress regarding a solution is rooted in a lack of consensus in identifying the problem.

A clearer understanding of resource sharing was built over time by importing notions from similar work in economics, where contention for scarce goods is well understood. The concept of pricing congestible resources was pondered as early as 1995 [MMV95]. The breakthrough in applying shadow pricing to a network would come through the work of Kelly [KMT98], who both proved that social welfare could be maximised if each pair of hosts were charged in accordance to the congestion they cause, and that under such a model self-interested behaviour would lead to a stable network under a small set of assumptions. Furthermore Kelly showed that such a scheme could be easily realized by charging a receiver according to the number of bytes marked as having experienced congestion through the use of

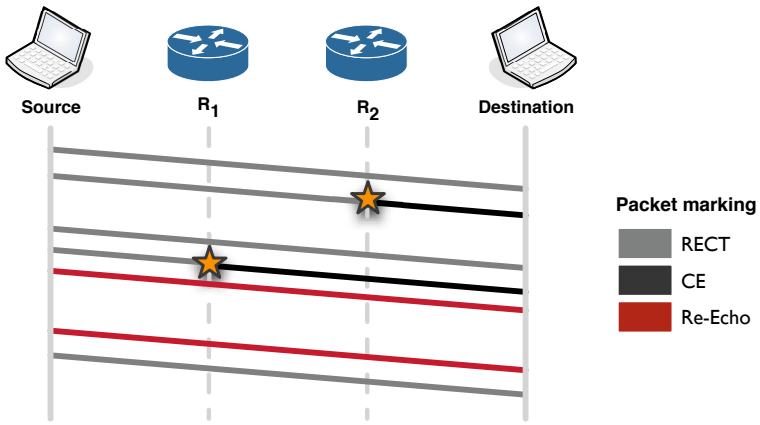


Figure 2.2: Simplified model of re-ECN.

ECN which was still undergoing standardization. Kelly's work proved a radical departure from conventional wisdom. By sharing capacity equally amongst flows TCP had been hard-wired with a single concept of fairness. That social welfare was shown to be maximized only when users were forced to weigh demand according to expected cost suggested that flow fairness was sub-optimal. As ever the notion of charging a user variable costs proved unpopular [Odl04], while charging the receiver instead of the sender proved impractical. Building on Kelly's work, Briscoe would resolve both shortcomings.

In [BJCG+05], Briscoe et al. introduce the concept of *re-feedback*. Re-feedback attempts to correct an information asymmetry in forwarding traffic: a provider often knows less about the quality of service it provides than the sender. Due to this asymmetry, Kelly correctly identified the receiver-end as the only point where the entirety of path congestion could be accounted for by the network. Information on congestion provided by ECN is fed back from the receiver to the sender in the transport header, and is therefore not visible to routers. Furthermore, the return path may differ from the forwarding path. By reintroducing this explicit feedback in the next outbound packet however, a sender can provide the network with information on the state of the forwarding path, albeit with one RTT of delay. Applying re-feedback to ECN lead to the specification of re-ECN [BJMS08]. In addition to information on congestion upstream, provided by ECN, re-ECN introduces the congestion over the entire path from the previous RTT in the IP header.

A simplified illustration of how re-ECN functions is displayed in figure 2.2. A source emits packets with the Re-ECN Capable Transport (RECT) code-point set. Upon detecting the onset of congestion, a congested resource which is ECN enabled may flag the packet as CE. A destination then feeds the explicit congestive feedback back to the sender. So far the system described merely follows the behaviour of a standard ECN system. Re-ECN differs from ECN in that in addition to adjusting its sending window to congestive feedback, it marks the following packet as a re-echo. In figure 2.2, this allows router R_1 to estimate the volume of downstream congestion, towards the destination, by simply subtracting the volume of CE marked bytes from the volume of re-echoed bytes. By subtracting the current upstream congestion from the previous full path congestion, re-ECN allows intermediate routers to estimate downstream congestion. Revealing downstream congestion to routers in turn allows senders, rather

than receivers, to be made accountable for the congestion they are expected to cause.

An important contribution stemming from work on re-ECN was to expose a wider networking community to a new goal within resource sharing, pushing the research agenda from flow-fairness towards cost-fairness [Bri07], and proposing congestion volume as the metric on which cost should be assessed. Variable congestive charges for users were avoided by policing sending rates according to a congestion allowance [JBM08]. Once such an allowance was exceeded, a sender's traffic would be shaped into compliance by the ingress policer. While protocols such as XCP or queuing disciplines such as Weighted Fair Queuing (WFQ) allowed bandwidth to be distributed differently amongst flows, the notion of a congestion allowance adds a temporal dimension to congestion management. By managing their allowance, hosts are provided an incentive to shift bulk traffic to off-peak times and may complete short flows more aggressively than by using existing TCP congestion control.

Work on re-ECN would evolve into the Congestion Exposure (CONEX) working group within the IETF [MABW09]. Remarkably for a protocol change affecting both the transport and network layers, congestion exposure would garner limited support from either community. Many providers did not identify congestion as a problem and those that did, such as Comcast, were able to get by developing localized solutions [BKL$^+$09]. Others may have felt uncomfortable in exposing such a sensitive metric to potential competitors. Within the transport community there was a natural reluctance to move away from flow-fairness [FA08]. Arguably the greatest impediment to deploying congestion exposure is the fact it relies on many small changes which together radically alter how resource sharing is performed on the Internet. Proving the overall framework is robust and reliable enough to work on a large scale is non trivial, and even minor issues such as the time scale over which a congestion allowance should be replenished have the potential to trigger significant operational ramifications.

2.2 Pooling across multiple paths: traffic balancing

Congestion management typically attempts to share end-to-end capacity for a given path. Between each source and destination however multiple routes may be available. This section reviews how different entities attempt to explore path diversity in order to perform resource pooling.

2.2.1 Traffic engineering

Providers routinely attempt to balance traffic across available network resources. On each provisioning cycle, operators try to adjust their infrastructure to cope with expected demand. Between cycles however demand may fluctuate considerably, either due to variations in traffic patterns or alterations in the customer base. Given the static nature of physical infrastructure, between provisioning cycles traffic can be *shaped* to avoid overloading the current path, *assigned* to alternate paths, or the path itself may be *reconfigured*. The set of tools by which an operator can optimize routing and balance traffic to achieve a desired allocation of traffic across a network is commonly referred to as Traffic Engineering (TE). Traffic engineering methods may be applied to attain different QoS objectives such as minimizing maximum link load, balancing load distribution or minimizing delay and typically vary in scope and timescale.

The scope of a TE method is either restricted to an *intradomain* setting, where traffic is optimized

within a single domain, or *interdomain* traffic, in which a network influences the ingress and egress through which traffic crosses its borders. Traditionally there has been a stronger focus on intradomain TE due to the greater control an operator has over its own resources. Additionally, the possibilities and limitations of a traffic engineering process are in large part determined by the underlying routing protocol. Traffic engineering was first adopted within Multi Protocol Label Switching (MPLS) networks and then adapted to intradomain routing protocols such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS). Interdomain traffic engineering methods encompassing the Border Gateway Protocol (BGP) have proved more convoluted.

The timescale over which TE is expected to adapt to traffic demand also varies. In *offline* TE, a Traffic Matrix (TM) is collected tracking demand between ingress and egress nodes over a fixed time period. This traffic matrix then serves as a forecast of future demand in the next iteration of a recurring routing optimization process. The periodicity of routing updates is typically in the order of weeks or months in order to ensure network stability. *Online* TE on the other hand does not require a prediction of future demand and operates entirely by adapting dynamically to traffic on a timescale of hours or minutes. Online methods have remained largely unexplored, in part due to the difficulty in making sure the dynamic system converges in the absence of a global view on traffic.

Offline intradomain TE is the most commonly used variant of traffic engineering, and originally gained momentum with the deployment of MPLS [RVC01]. In MPLS networks, packets are forwarded according to labels rather than addresses. At each ingress Label Switching Router (LSR), a packet is encapsulated and assigned to a Forwarding Equivalence Class (FEC). The resulting label is then used to forward the packet along a Label Switched Path (LSP) towards the egress LSR. While there are practical limits in how many paths can be maintained and non-negligible overheads in setting up an LSP, the ability to forward packets explicitly along arbitrary routes was instrumental in initial attempts at effective traffic engineering [XHBN00, WW99].

In contrast, IP-based TE solutions are more scalable but less flexible. Traffic engineering was first applied to Interior Gateway Protocols (IGPs) such as OSPF by Fortz et al. [FT00, FT02], who proposed setting link weights according to traffic demand in order to attain TE objectives. Theoretical work would further prove that any arbitrary set of loop-free paths could be transformed into shortest paths through a given set of weights [WWZ01]. Unlike MPLS methods however, traffic cannot be unevenly split natively in IP traffic engineering. To do so requires the use of Equal Cost Multipath (ECMP), which balances traffic equally across next hop routers for paths with the same cost. Originally ECMP was specified to split traffic on a per-packet basis, which may lead to out-of-order delivery and affect transport protocols as a result [TH00]. This approach has since been obsoleted and replaced by splitting traffic on a per-flow basis [FGL+00].

Online intradomain TE proposals built on their offline counterpart and fulfilled a desire for controlling traffic at a finer granularity. Multipath Adaptive TE (MATE) [EJLW02] proposes splitting traffic dynamically over precomputed LSPs. By using probes, MATE continuously monitors packet delay and loss over each path between ingress and egress LSR in order to adjust the ratio of traffic to be split at

the ingress. TE with XCP (TeXCP) [KKDC05] improves on MATE by emitting probes which retrieve explicit feedback from intermediate TeXCP-aware routers in a manner similar to XCP. Furthermore, TeXCP distributes load using Flowlet Aware Routing Engine (FLARE) [SKK04] which balances traffic by *flowlet* rather than flow. Given the bursty nature of TCP, traffic splitting may be performed on a smaller scale than a flow so long as there is no risk of causing packet reordering. A flowlet is therefore defined as a sequence of packets with an inter-arrival time shorter than the difference in delay between potential paths. Balancing by flowlet allows FLARE to achieve not only greater efficiency than flow variants of ECMP, but also maintain less state as flowlets expire more rapidly than flows.

A second approach to online intradomain TE is to dynamically optimize routes in accordance to demand. In MPLS networks this may however cause LSP interference [KL00], in which excessive paths crossing the same critical link are set up resulting in congestion. For IP-based TE changing link weights dynamically is possible, but strongly discouraged due to potential instability during the convergence process [LMJ98].

Interdomain TE has received less attention than intradomain traffic engineering and can be further categorized according to whether it is applicable to *outbound* traffic or *inbound* traffic. The methods available in either case are strongly dependent on the path selection process [QPS+03] in BGP, the prevalent interdomain routing protocol.

Outbound TE is most often performed by manipulating the local preference (Local_pref) attribute of a route for each egress router, with the highest preference assigned to the preferred egress. For sufficiently tied outbound routes, in which amongst other metrics the Local_pref and Autonomous System (AS) path length are equal, the lowest IGP weight can be used as a tie breaker. This form of *hot potato* routing, which offloads traffic to another domain as quickly as possible, can be manipulated to achieve TE objectives through the appropriate setting of IGP weights. Explicit interdomain routing is possible using MPLS if domains agree on cross border path establishment, but remains rare. Proposals for outbound, interdomain TE typically attempt to minimize transit costs and delay [UB04, GQX+04] or provision QoS requirements [HFP+05].

Research on inbound TE has been sparse as the route inbound traffic takes can only be influenced rather than controlled. Mechanisms for affecting another operators choice in outbound path include selectively advertising routes from specific ingress links, prepending the announced AS path with multiple instances of the same AS to discourage its use [CL05], or manipulating the Multi-Exit Discriminator (MED) or community attribute to signal preferences to other domains [QTUB04].

As an afterthought in the Internet architecture, traditional traffic engineering suffers from being both hidden from and oblivious to higher layers. With no indication as to how TE operates, hosts and other networks will continually attempt to balance traffic according to their own needs potentially subverting existing routing optimizations. With no understanding of how or why hosts balance traffic, TE may be futile in outpacing dynamic traffic patterns [HCR06]. As a result of this architectural opaqueness, TE can neither adapt too quickly or quickly enough. This double bind has left traditional TE confined to operating over extended time scales largely out of fear of causing disruption [LMJ98]. That traffic

engineering is widely used in spite of being poorly understood only reinforces that it arises from a valid concern, but whether routing optimizations are the most appropriate way of addressing such concerns remains open.

2.2.2 Resilient routing

The previous section detailed how routing is burdened with the responsibility for balancing traffic efficiently through the use of traffic engineering. A further concern often placed on the routing architecture is in providing resilience. A routing protocol should by design reliably recover from failures, but the degree to which such reliability can be assured depends on the time taken to *detect* a failure and then reach a consensus and *converge* towards a new operating state which bypasses the failure.

Failure detection in link state protocols depends on the frequency of probes confirming network adjacency, typically referred to as Hello packets. In OSPF this periodicity is configured through the HelloInterval variable, and a failure detection event is triggered once the RouterDeadInterval has been exceeded with no reply from a neighbour to Hello probes. Traditionally the HelloInterval has been set to tens of seconds in default configurations. Reducing this interval results in faster failure detection [GRF03] but doing so excessively may also lead to spurious failure events resulting in routing instability. In IS-IS the HelloInterval is set at a second granularity and a failure is detected upon the loss of three Hello packets, placing the lowerbound on failure detection at 3 seconds.

The deployment of real time applications with harder constraints on reliability coupled with better failure detection methods embedded in linecards have provided both the motivation and the means for achieving sub-second recovery within IGP networks [FFEB05]. Even with reduced recovery times, the transient effects of routing changes can still disrupt the forwarding path. Under such cases the Fast Re-Route (FRR) framework, applicable with minor changes to both IP [SB10] and MPLS [PSA05] based routing protocols, provides repair paths which may be used between the detection of a failure and the convergence of the routing process.

A commonly employed technique for constructing repair paths is to pre-compute Loop-Free Alternate (LFA) next hops [TAC$^+$08]. Upon detecting a failure of the default next hop n_d, node s may forward a packet towards destination d through backup neighbour n_b in a loop-free manner by verifying the following condition:

$$cost\,(n_b, d) < cost\,(n_b, s) + cost\,(s, d) \tag{2.6}$$

that is, the cost of routing the packet from the alternate next hop to the destination is less than the packet being looped back to the source en route to the destination. While this ensures link protection, failures often occur at nodes. To protect against node failure, a next hop must additionally satisfy:

$$cost\,(n_d, d) < cost\,(n_b, n_d) + cost\,(n_d, d) \tag{2.7}$$

that is, the cost of routing the packet from the alternate next hop to the destination is less than routing the packet from alternate next hop to the destination via the default next hop. A final condition must be

met to avoid routing loops in the presence of multiple failures, ensuring that packets progress towards the destination at all times:

$$cost\,(n_b, d) < cost\,(s, d) \tag{2.8}$$

LFA incurs low computational and memory overhead and requires no changes to the forwarding plane, making it a practical choice for bypassing most failures. Further proposals [Atl06, BFPS07, SPB11] would provide coverage for increasingly complex failure scenarios at higher implementation cost.

A long-standing concern with resilient routing is the potential for cascading failures, in which detouring leads to overload at a separate point in the network and potentially triggers more failures. As with traffic engineering the use of multi-topology routing can assist in distributing load through the availability of different routing planes. This concept is explored in Multiple Routing Configurations (MRC) in which successive routing configurations are pre-computed by isolating nodes and links, thereby covering all single failure scenarios. Upon detecting a failure, the plane on which a packet is forwarded is switched to the configuration where the failing resource is isolated. Overall load distribution is improved through the offline calculation of alternate routes, but guaranteeing recovery from all single failures may not be scalable for some topologies.

Most research in network resilience has been limited to within a single domain. In part, this is due to the perception that upon leaving a domain an operator is no longer responsible for verifying traffic is delivered. Unfortunately the end-to-end nature of traffic often leads to misattribution of blame upon ISPs in particular, who share the burden of third-party failings through customer support. More importantly however, the lack of deployed multipath alternatives for BGP reduces the flexibility with which detouring can be performed. A practical solution in this space seems inevitable however: in addition to many proposals enabling multipath interdomain routing [Yan03, XR06, MEFV08, GGSS09], the emergence of alternate, deployable routing architectures to address scalability concerns such as Identifier Locator Naming Protocol (ILNP) [ABH09] and Locator/Identifier Separation Protocol (LISP) [LFFM12] are undergoing standardization. Whether BGP is extended or locator and identifier functions become decoupled from existing addressing, the ability to manage interdomain traffic resiliently is likely to improve.

2.2.3 Higher layer approaches

Concurrently to traffic balancing within the network, higher layers have developed different techniques to manage traffic according to their own needs.

The introduction of concerted attempts to balance traffic from the host would arise with the advent of the Web. HTTP flows were as short as they were numerous and represented a new class of traffic which had not been foreseen when designing TCP [Day10]. Scaling infrastructure to cope with thousands of connections robustly would require spreading load efficiently across servers. To attain seamless load balancing content providers resorted to an existing layer of indirection: the Domain Name System (DNS) [Moc87]. In replying to a query, a DNS server can balance traffic by providing a different answer from a

pool of available servers [Bri95]. Merely iterating through the list in round robin fashion is sufficient to balance traffic coarsely, but efficiency may often be affected since an answer can be cached and re-used by a potentially large set of clients. To provide increased fault-tolerance DNS load balancers quickly evolved to return results in accordance to server load and availability. More recently, the widespread geographic distribution of hosting infrastructure has reasserted the importance of DNS in managing traffic and optimizing delay in particular [AMSU11].

By explicitly selecting a content source, DNS load balancing permits hosts to implicitly change the network path. The ability for hosts to explicitly change the network path had originally been provisioned in the IP specification through source routing, but was quickly obsoleted. Source routing was initially intended to both facilitate network debugging and overcome inherent shortcomings in contemporary routing protocols [Sun77]. Neither would justify maintaining source routing in the long term as the Internet expanded and became increasingly commercial. In its *strict* form, source routing required prior knowledge of every router along a path which was infeasible given the sheer scale of the topology. In its *loose* form, a host could define intermediate routers a packet would traverse. While more practical than strict source routing, loose source routing as defined in IP was inherently insecure as the ability to spoof IP addresses and establish routing loops could lead to amplification attacks from malicious hosts. Critically, operators disliked both forms of source routing, neither willing to share operational details of their network nor keen in supporting a mechanism which allowed users to subvert local routing policies. The combination of all of the above would eventually lead to the deprecation of the Loose Source and Record Route (LSRR) option.

Despite having been disabled from the current architecture, source routing has been frequently revisited in research. Traditionally source routing has been viewed as a host-centric solution, allowing senders to select better performing paths or circumvent traffic discrimination [MZPP08, DMG+10]. More recent research however has attempted to recast source routing as a potential asset to an ISP.

Perhaps the greatest benefit source routing offers operators is in reducing the impact of resilience on the routing architecture. In Scalable One-hop Source Routing (SOSR) [GMG+04], Gummadi et al. investigate the usefulness of one-hop source routing in order to improve reliability. Through the use of active measurements over a week to measure path availability, Gummadi estimated an average path availability for servers at 99.6%, and for broadband hosts at 94.4%. Gummadi then implemented a simple "random-4" hop selection which attempts to detour around failures by selecting four random intermediate nodes from a pool of available choices, managing to recover flows in 56% of possible cases. This study provided two important results. Firstly, in localizing failures the study asserted that for popular servers failures were more likely to occur in the core of the network than close to the last hop. This is likely to correspond to the majority of Internet traffic as content becomes increasingly consolidated [LIJM+10a]. Secondly, by investigating different policies for the selection of the detour hop Gummadi further corroborates an intuitive result: that most benefit can be derived from a small number of alternatives.

Detour routing had previously been investigated in the context of overlay networks such as Resilient

Overlay Routing (RON) [ABKM02], but implementing such a function at the network layer would require a more opaque form of source routing in order to appeal to network operators. In [YW06], Yang and Wetherall propose an incrementally deployable routing deflection scheme through the use of tagging. The main contribution resides in the separation of how a deflection is constructed from how a deflection is invoked. A router constructs a *deflection set* composed of viable routes to a given destination, in a similar manner to LFA, and then forwards packets along deflections in accordance to a tag in the network header. Hosts are then responsible for setting this tag and may modify it at any time. The resulting scheme can offer much of the potential hinted in SOSR, but without the requirement for networks to reveal operational information. Hosts merely manipulate their tag until they achieve a desired outcome.

The fundamental tussle between end-user intentions and ISP control would further be investigated in [LJC08]. In analysing "user-directed" routing Laskowski et al. claim that such a tussle can be resolved given a flexible enough payment system. While no suggestion is made on what form such a payment system would take, a market-based approach is shown to allow ISPs to induce any feasible traffic pattern. In the absence of a suitable solution for source routing, applications have improved path performance and availability implicitly through the use of overlay architectures. Popular peer-to-peer applications such as Skype or Bittorrent already derive many of the benefits of pooling traffic across multiple paths, but do so in a manner which may be detrimental to competing traffic. P2P applications ensure path diversity by establishing flows to multiple peers. With TCP sharing bandwidth equally amongst flows, an application making use of multiple flows may gain a significant advantage over traffic flowing through a shared bottleneck. In addition to consuming a disproportionate amount of bandwidth, in consistently saturating links such applications may induce significant delays. Solutions for attenuating the impact of such traffic would take many forms [PC09], from traffic shaping to weighted congestion control, but in framing a multipath overlay atop of a single path transport layer a pertinent question would emerge: what form should a multipath-aware transport layer take in an Internet dominated by TCP?

Multihoming would provide the initial motivation for developing a socket abstraction for the concurrent use of multiple paths [Hui95, HS02, IAS06]. These early attempts would mostly focus on the protocol changes required to achieve a robust and efficient multipath transport service. The implication of such flows on fairness and the impact of multipath transport on overall system stability would first be explored in theoretical work by Kelly and Voice [KV05] who would use a fluid-flow model to demonstrate traffic balancing can be performed at end-hosts in a stable manner on the same timescale as rate control. In evaluating an end-to-end algorithm for joint routing and rate control, the authors conclude that while the network layer is able to provide *structural* information on routing, dynamic routing is more naturally performed by the transport layer. Work by Key, Massoulié and Towsley [KMT07] would corroborate these results. The authors show that the use of parallel connections, a form of *uncoordinated control*, can lead to inefficient equilibria in the presence of RTT bias which afflicts most TCP variants. Importantly, the use of a *coordinated* congestion controller, which actively shifts load between a set of paths in accordance to their state, can result in a welfare maximising state even if with RTT bias. These theoretical advances would take shape in Multipath TCP (MPTCP) [WRGH11], currently undergoing

standardization.

While MPTCP promises many of the benefits predicted by Kelly and Voice, some practical issues remain unresolved. Firstly, access to path diversity is ensured through the use of multihoming. This has the unfortunate side-effect of further overloading the IP address as a *path identifier* in addition to locator and host identifier. Furthermore, end-host multihoming, while prevalent in mobile devices, is not commonly used. Secondly, it is unclear what proportion of flows can resort to MPTCP. The combination of overhead in setting up multiple sub-flows and time required to adapt the sending rate to the channel capacity limits the applicability of MPTCP to long transfers. The majority of flows are too short to use MPTCP, but could derive much benefit from using multiple paths, particularly in terms of resilience.

2.2.4 Rethinking traffic management

The proliferation of highly replicated content across P2P systems, Content Distribution Networks (CDNs) and One-click Hosting (OCH) services has motivated research in alternative forms of traffic management. By allowing users to download the same content from multiple sources, networks can explore opportunities for load balancing by influencing from *where* or *when* content is retrieved.

Exploiting the prevalence of content replication in P2P applications for the purpose of reducing ISP costs was first proposed in Provider portal for applications (P4P) [XYK+08]. P4P proposed deploying portals operated by network providers through which peers could query a collection of interfaces detailing network policy, costs or capabilities from a provider's perspective. Given this information, a host could then select a set of remote peers which would minimize the impact of traffic in the network. For ISPs, P4P provides a means of signalling costs. P2P applications on the other hand benefit from not being discriminated against by being sociable, and not having to probe the network to derive network topology or status. P4P would evolve into Application Layer Traffic Optimization (ALTO) and undergo standardization within the IETF. Similar approaches for reducing cross ISP traffic would be explored in ONO [CB08], which collects information implicitly available in CDNs to establish proximity of peers.

Research in harnessing content replication has followed the shift in content distribution from P2P software to CDNs hosting. In [AMSU11] the authors investigate CDN and hosting infrastructures to establish the proportion of content unique to a single provider and show that some hosting infrastructures have as much as 93% of their content available elsewhere. This property is exploited in [PFS+12], in which the authors propose Content-aware Traffic Engineering (CaTE), which allows ISPs to take advantage of content available in multiple locations to reduce link utilisation. In CaTE, an ISP influences where a content request is redirected to by intercepting DNS requests and selecting a server according to local objectives. Interestingly, CaTE may still present advantages to content providers, since ISPs have more accurate information on the location of their customers and tend to choose shorter routes.

In parallel, multiple proposals have explored the potential for shifting delay-tolerant traffic to off-peak hours. In [LR08] the authors describe a mechanism that offers users higher bandwidth off-peak if they deliberately delay some of their traffic. Further contributions [JWHC11, CLRS10] represent similar attempts to shift traffic in time by providing incentives to users.

Traditional traffic engineering manipulates routing weights given an expected demand. Such an

approach was acceptable when Internet traffic revolved around communicating endpoints - the source and destination of a flow were assumed to be immutable. The Internet has long shifted towards distributing *content*, and with it become characterized by fluid traffic patterns which adapt and react to changes in routing. Under such conditions, it is more flexible for operators to subvert TE and adjust demand given a set of routes in order to attain an intended QoS objective. CaTE and ALTO attempt to reflect network interests in the selection of an end-to-end path and focus on a small set of sources of high volumes of traffic: P2P applications and CDN downloads. While this may currently be effective in load balancing traffic within a network, such application specific approaches may become ineffective in the long run as traffic patterns continue to change with the introduction of novel, disruptive applications.

Chapter 3

A mutualistic resource pooling architecture

While the Internet has become evermore interconnected, exploring path diversity has been relegated to an afterthought in an architecture modelled around assumptions that no longer stand. Single-path forwarding as a paradigm arose not as a guiding principle, but as a natural aversion towards increasing both the complexity and cost of a resource starved network. Engineering for scarcity has propelled the Internet to an unprecedented scale, but nagging issues arise when what was otherwise scarce becomes plentiful. Protocols designed to be bit conservative at the expense of latency have become technological anachronisms as bandwidth costs continue to plummet. Similarly, the notion of a router as a device merely capable of forwarding packets has long been obsolete as Moore's law continues to pave the way for greater functionality within the network. Network Address Translator (NAT), DPI or Performance Enhancing Proxy (PEP) middleboxes are all examples that when it comes to drawing a boundary between network and transport, the line begins to blur [FI08].

Furthermore, parallelism seems to be a dominant trend at every level of the Internet architecture as a cost-effective means of increasing both performance and robustness. At the inter-domain level, the AS graph is becoming flatter and more highly interconnected [HFU+10]. Within domains, the sheer complexity of managing paths has led to the streamlined design and deployment of MPLS [RVC01], implementing a fully fledged layer in its own right. At the edges, the rise in multi-homing continues to increase the strain on an already overloaded routing architecture. Even within network components, parallelism is such that packet re-ordering can no longer be considered pathological [BPS99].

Given these trends, one would expect the ability to pool traffic across such emergent path diversity to have become a network primitive. In reality, each stakeholder in the Internet architecture seems to balance traffic according to their needs while attempting to remain inconspicuous to others. At best, this interaction between stakeholders can be seen as a form of commensalism, where one entity can extract benefits while others remain unaffected. At worst, the competitive nature of the tussle [CWSB05] that ensues can spiral into a situation where few profit.

This chapter investigates the nature of this antagonism between network and endpoints and reflects on how the Internet can accommodate the needs of both through the use of PREFLEX, a proposed architecture for balancing congestion which promotes mutual cooperation between end-hosts and edge network providers.

3.1 Resolving the tussle

The ability to evolve beyond single path forwarding has often been misdiagnosed primarily as a routing challenge. The subject is frequently revisited with varying approaches [Sun77, Yan03, YW06, GGSS09]. Despite this, multipath routing has remained elusive for end-hosts. The common trait all these proposals share is a failure to identify the tussle over resource control as a significant obstacle in moving towards the use of multiple concurrent paths.

Given an explicit goal of the Internet was to accommodate a variety of networks, very few assumptions could be made on the basic functionality they could support. The Internet architecture therefore placed resource control at the edges, in what can be viewed as an instantiation of the end-to-end principle [SRC84]. This represented a fundamental paradigm shift, ultimately conferring the scalability which fuelled the growth of the Internet. While unilateral control of a network resource by hosts was already polemic in an academic research network, with the rise of the commercial Internet this notion has slowly been set aside by stakeholders intent on exerting control over their own networks.

Network operators have now become accustomed to inspect, shape and throttle traffic in an attempt to override resource sharing as implicitly performed by TCP. A common cause for such behaviour could derive from the perceived free riding made possible by TCP, whereby a minority of users can gain an disproportionate amount of bandwidth, with detrimental effects for the majority of users. In a broader sense, networks attempt to reflect their own objectives and concerns. Because this was not contemplated when designing the existing resource sharing model, subsequent violations of the end-to-end principle say more about the limitations of the current architecture than the ill intent of the perpetrators.

Nowhere is this more apparent than in traffic engineering. Network operators rely heavily on TE to balance utilisation over long time scales in an attempt to reduce costs by making efficient use of available paths. Since information at the network layer is limited, TE typically optimizes locally for the wrong metric – utilization – in detriment of the congestion it may be causing elsewhere. Additionally, this optimization is typically executed offline, and re-computed over long time scales to minimize the impact to higher layers and ensure stability. The limiting assumption is that traffic patterns are exogenous. In reality, hosts will often find means of adapting to network conditions, such as establishing overlay networks. This resulting shift in behaviour may in turn conflict with the concurrent traffic engineering process, which will have to readjust to a substantially different traffic matrix in a next iteration. Traditional traffic engineering mechanisms can therefore neither adapt too often out of fear of disrupting transport protocols, nor adapt often enough in order to adequately react to changes in traffic. As a result, there is still considerable space for improvement in existing traffic engineering:

The problem of optimizing routes given an expected demand has been solved. Solutions for offline traffic engineering [FT00, WW99, WWZ01] are often touted to be optimal despite relying on approximate inputs, but for the most part the resulting routing optimizations are *good enough*. What remains to be solved is how traffic engineering can deal with deviations from the predicted traffic demand. For any form of online traffic engineering, the act of balancing traffic on-the-fly over a multipath routing architecture is perceived as more practical than attempting to design a

routing substrate which adapts to traffic demand. Given a stable set of routes, how should traffic be balanced?

At a small enough timescale, traffic engineering can replace resilient routing entirely. For any sufficiently reactive TE process, a path failure will lead to a shift of traffic onto alternate paths. Even online approaches such as MATE [EJLW02] and TeXCP [KKDC05] have hesitated to be computed at a sub-minute granularity however, and in many cases hosts are more capable of shifting traffic robustly given access to underlying path diversity. How can traffic engineering be designed to offer better resilience than routing protocols, and in what cases can it delegate such functionality to hosts?

The distinction between intradomain and interdomain TE is a hindrance. The duality emerges due to traditional traffic engineering being tightly coupled with the underlying routing protocol. All traffic engineering within a single domain however must target the same objectives or risk instability. A further source of instability of TE methods is in taking local decisions without taking into account the end-to-end ramifications. How should traffic engineering methods be unified, and how can timely end-to-end information on traffic be collected in a scalable fashion within the network?

The network has appropriated flow semantics from the transport layer. Balancing traffic per-packet has been replaced by a per-flow granularity, further ingraining transport behaviour in the network. In part this reflects the changing nature of the IP model, which now expects in-order delivery from the network [Tha10]. FLARE [SKK04] illustrates that maintaining in-order delivery need not span the entirety of a flow, but maintains the notion of the flowlet internal to the network. What are the potential benefits of explicitly decoupling the notion of a transport flow from a network flowlet?

In stark contrast with traffic engineering, the interest in the use of congestion control to balance traffic across paths has gained significant traction, particularly in the wake of seminal contributions [KMT07, KV05] which provide the theoretical basis for much of the standardization effort behind MPTCP [WHB08]. MPTCP provides the means for stable traffic balancing from the transport layer. This alone however is unlikely to overcome significant architectural shortcomings:

Path diversity is in the network. Given networks are already concerned with TCP's ability to share bandwidth in its single path incarnation, it remains unlikely ISPs will consider making path diversity visible to end-hosts. This opaqueness currently restricts deployment of MPTCP to multihomed hosts. Experiments with one-hop source routing [GMG+04] indicate that a small set of network paths can provide the majority of the benefit in terms of resilience. While multihoming may become popular amongst hosts, it is already a requirement amongst providers. Diversity is in the network, how can it be pushed outwards and made available to end-hosts?

Traffic is best balanced from the edge, but may conflict with network objectives. Work by Kelly and Voice [KV05] indicates load balancing can be more effectively performed at the transport

layer in order to maximize social welfare. However this fails to take into account the requirements of network operators, who are saddled with costs which bear little resemblance to the congestion pricing schemes advocated by Kelly. While TE methods are crude, they address valid concerns and accommodating these mechanisms within the Internet architecture is essential in order to minimize conflict with end-to-end resource pooling. Re-ECN [BJMS08] demonstrated the value of congestion as a metric within the network, but its operation is too tightly coupled to ECN which is itself not widely deployed. Additionally, retrofitting accountability into the Internet seems marred in technical pitfalls. If the constraints on providing accurate accountability are relaxed, can some form of congestion exposure provide useful feedback on host preferences to the network?

Most flows may not benefit from MPTCP. For short flows, the overhead incurred in multiple subflow establishment is excessive. Furthermore, for long flows with bursty behaviour, such as rate-limited video streaming [RLL+11], MPTCP may not have sufficient time to pool bandwidth across multiple paths efficiently. The transport layer requires probing the network for every connection. With no prior information or limited time to collect information on network state, the full benefits of multiple paths for resilience in particular are unlikely to be harnessed. In such cases, how can the network complement, rather than necessarily override, transport balancing?

Neither congestion control or traffic engineering alone seem fully capable of bridging the divide between networks and end-hosts. The discussion around the relative merits of both is often manichaean and erroneously simplified as a conflict between advocates and opposers of the end-to-end principle. This entirely misses the point. *The concern should not revolve around whether an approach is right or wrong, but whether it is applicable within a given context or not.* The recognition of the commercial network as a fundamental stakeholder is intrinsic to the evolvability of the current architecture. In the absence of such recognition, the gulf between the perception of how the Internet should behave and how it functions in practice will only widen. Traffic engineering and congestion control represent an explicit duality – underlying both should lay a unifying architecture allowing either to evolve independently while not foregoing cooperation.

Recent research in resource sharing has suggested that much of the misalignment between network and transport derives from the lack of accountability for congestion [Bri07]. While previous work had modelled and analysed the broken incentive structure subjacent to forwarding traffic from an economic perspective, work on re-feedback and congestion exposure [BJCG+05] pioneered a practical means of alleviating the tussle surrounding resource sharing. In particular, congestion exposure advocates the use of congestion volume, rather than throughput or traffic volume, as the by-product by which the impact of traffic should be assessed. Building upon this approach, the next section presents PREFLEX, a joint, mutualistic architecture for congestion control and traffic engineering.

3.2 PREFLEX

The PREFLEX architecture can be split into two independent components. At the network, a mechanism for Path Re-Feedback (PREF) is defined, whereby stub domains can signal a preferred path to end-hosts

according to local policy or perceived path quality. At the end-hosts, a transport agnostic protocol for Loss Exposure (LEX) is used, which explicitly marks packets within a flow in order to signal path loss back to the network.

While functionally separate, in practice both components work in tandem. The use of loss exposure, while executed by hosts, provides network operators with feedback on end-to-end path loss. Conversely, with path re-feedback hosts are allowed access to paths selected by the network. Together, PREFLEX bridges the divide between network and transport layers and facilitates balancing by congestion, rather than necessarily load, over the multiple paths typically available solely to edge networks.

3.2.1 Loss Exposure

LEX is proposed as a simple protocol for revealing loss, which not only borrows heavily from re-ECN [BJMS08], a protocol for congestion exposure, but which can coexist and serve as a stepping stone for the deployment of the latter. Revealing information currently confined to the transport layer down to the network both reduces the need for the network to inspect higher level protocol headers in order to re-distribute bandwidth differently and corrects the information asymmetry that currently afflicts networks, who know less about the quality of service they provide than their customers.

The first change proposed for LEX is to have end-hosts mark, at the network layer, packets belonging to flows where feedback has not been established. This typically corresponds to the first packet exchange in a flow, such as SYN packets in TCP, but may also include the first packet after a significant idle period, a keep-alive packet or a renewed attempt at a retransmission after successive timeouts in the case of network failure. Within LEX, as with re-ECN, all such packets where a flow does not have accurate and timely information on network conditions should be labelled as Feedback Not Established (FNE), with the IP header being marked accordingly.

The signalling of such packets has many practical implications. For one, from simply inspecting the IP header, networks are made aware of the first of a succession of similar packets, which poses significant advantages in allocating state in middleboxes, whether it be to perform admission control, policing or traffic shaping. All of the above are possible by inspecting TCP, but this makes apparent an architectural illusion: that a connectionless layer should be oblivious to connection setup. Explicitly providing such information at the IP layer alleviates in some measure the need for consistent violation of layering by network equipment, or hopefully circumscribes such practices to a small subset of packets.

Additionally, the concept of a transport flow, which establishes an association between two endpoints, is decoupled from the concept of a network flow, which will henceforth be referred to as a flowlet [SKK04]. A flowlet is defined as a stream of packets which end-hosts expect to follow the same network path. The same transport flow may be composed of a single flowlet, parallel flowlets, or a succession of different flowlets. This feature is particularly advantageous for balancing traffic as flowlets provide a finer granularity than existing flows, as well as allowing flows to quickly switch path without breaking the transport session.

Once feedback has been established, hosts adjust their sending rate in response to implicit congestive signals such as delay or packet loss, or explicit signals such as ECN. Protocols for congestion

CODE POINT	MEANING
Not-LECT	Not Loss Exposure Capable Transport
LECT	Loss Exposure Capable Transport
LEx	Loss Experienced
FNE	Feedback Not Established

Table 3.1: LEX code points and description.

exposure, such as re-ECN, mark outgoing packets according to the explicit congestion marking received from the network. As such, IP packets would carry two congestion markings. The first indicating the congestion experienced so far and the second indicating the end-to-end congestion experienced by the host in the previous RTT. With this re-feedback of congestion markings, networks are able to estimate rest-of-path congestion, which is an important metric for keeping customers accountable for the congestion they cause and providers accountable for the services they offer.

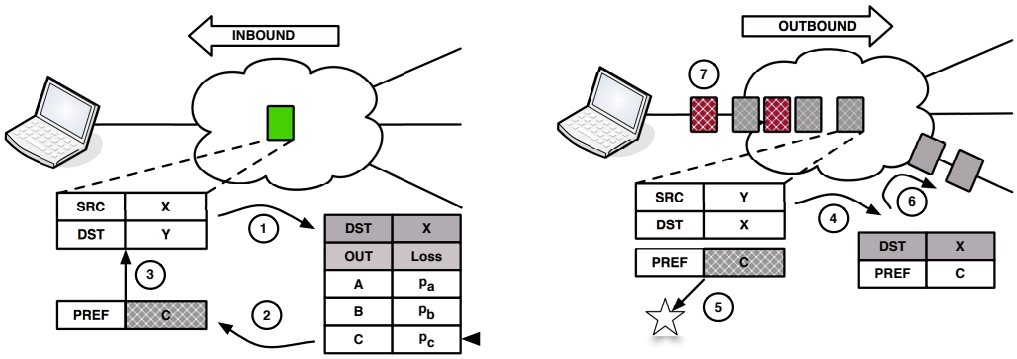
LEX specifies a simplified form of congestion exposure which uses the implicit information contained in losses as opposed to relying on the widespread deployment of congestion notification. Where packet loss does arise, LEX requires that hosts mark their respective retransmits with a LEX code point. The drawback of this approach is that only the end-to-end congestion can be estimated from a stream of packets, which implies that traffic can only be reliably aggregated close to the source, and effectively policed close to the receiver. Since the focus of PREFLEX is balancing congestion at a stub domain however, this limitation is not significant.

If run as a complement of re-ECN, three of the four code points in table 3.1 are potentially shared, in which case only the loss experienced code point has to be added to the re-ECN specification. For routers along the path, an accurate estimate of the end-to-end path loss can be obtained by simply dividing the sum of bytes marked with the loss experienced code point, by the total traffic marked as either Loss Exposure Capable Transport (LECT) or LEX. Additionally, one could envision a preferential dropping mechanism which prioritizes retransmits.

3.2.2 Path re-feedback

For networks, the most significant hurdle in adopting multiple paths for a single destination is not the path selection process, but rather the difficulty in assigning packets to paths. Since balancing traffic at a packet granularity has severe repercussions for the transport layer, network operators have typically resorted to splitting traffic by destination prefix. Increasingly, networks have also been able to afford the cost of keeping flow state in an attempt to balance traffic at a finer granularity.

Neither of these approaches are strictly necessary in a mutualistic architecture – one in which a division of responsibility between host and network is performed for the benefit of both. Since hosts are made aware of the path packets take, flow state can be pushed outwards, placing the responsibility for assigning packets to paths at the endpoints. This enables hosts to benefit from existing path diversity. Networks then only need to perform path selection according to local policy and pass the information



(a) Inbound FNE packet

(b) Outbound traffic

Figure 3.1: PREFLEX architecture.

onto the end-host. This allows finer control over traffic in a scalable manner.

For this purpose, FNE packets, as defined in LEX, are used to act as network triggers for path selection. An ISP or stub domain, upon detecting an incoming FNE packet, selects a preferred outgoing path based on the reverse lookup of the source address, and marks the packet with a path identifier. For IP Version 4 (IPv4), a possible location for such marking to occur could be within the Differentiated Services (DS) field, where a set of code points are reserved for local use. On receiving an FNE packet containing a path identifier, a sender should tag all subsequent packets in the flowlet using the same identifier in order to ensure it will traverse the selected egress at the edge domain.

This behaviour is exemplified in figure 3.1. In PREF the network selects a preferred outgoing path for each incoming FNE packet (figure 3.1a). Upon receiving the first packet of a flowlet, the PREFLEX aware router performs a reverse lookup on the source address (step 1) and selects a path according to the perceived performance (2). The router then associates the chosen path identifier to the packet and forwards the packet toward the host (3). The treatment of outbound traffic by PREFLEX is illustrated in figure 3.1b. The host, having received indication of the preferred path, tags all subsequent traffic with the given path identifier. As the PREFLEX aware router receives this marked traffic it updates statistics associated to path, aggregating loss for each destination prefix (4). It then discards the path identifier (5) and forwards traffic along the appropriate path (6).

A subtle implication of triggering path selection based on incoming packets, rather than resorting to out-of-band signalling for example, is that path selection becomes receiver driven. The responsibility for defining a strategy on when and how often to attempt a path request lays firmly with the stakeholder who extracts the most benefit. For example, a receiver valuing flow completion time may decide to minimize the amount of FNE packets it sends in order to reduce network processing. Conversely, a receiver valuing resilience or efficiency may emit additional FNE packets in the hopes that the sender may acquire more network paths. The flip side is that because FNE packets require additional network intervention, whether for selecting a new path or setting up state, networks may rate limit the amount of FNE packets they receive in order to protect themselves from overload. This is the current line of thinking with re-ECN, where FNE packets are used to set state in congestion policers.

3.3 Closing the loop

This chapter broadly described an architecture which shares the responsibility for resource pooling between end-hosts and edge networks, but does not explicitly dictate an outcome. PREFLEX has been designed to take into account the inevitable tussle which will occur between both, and envisages use cases where control over resource pooling could feasibly shift entirely in one direction or the other.

At its most liberal, PREFLEX enables resource pooling to be entirely performed by end-hosts. At its most conservative, PREFLEX affords edge network providers more fine-grained control over traffic than before. Between either extreme, the resulting mutualistic architecture offers greater transparency, control and robustness by realigning the interface between network and transport in order to accommodate the needs of both.

In order to make PREFLEX deployable, the scope of the architecture has been restricted to stub domains. While this necessarily reduces the amount of path diversity which can be explored, there are compelling reasons to abdicate some flexibility in favour of a more practical solution:

Most benefit derives from a limited set of paths. Work on source routing [GMG+04, YW06] suggests most improvement in resilience can be extracted from a small set of deflections. Stub domains such as ISPs and enterprise networks are naturally multi-homed, and can provide reasonable path diversity given an adequate selection of providers. Expecting the establishment of cross-domain paths has been a pitfall for previous research in QoS in particular and is marred by difficulties in providing incentives for all parties involved.

Stub domains are most aligned with user interests. For enterprise, academic and content distribution networks, end-hosts are typically managed by the same entity which operates the network. For ISPs, there is a binding contract between end-users and provider. In either case, the stub domain not only benefits from not deteriorating the quality of service provided to hosts locally, but also has an interest in making sure end-to-end traffic is unaffected by outages or degraded performance. The inability to reach a website is often misdiagnosed by users as a failure of the stub domain, rather than the intervening network path or the remote host. As a result, the burden of remote failures is often placed on local customer support. PREFLEX allows resources not only to be shared more efficiently, but also potentially reduces the impact of remote network events on stub domains.

The Internet topology is flattening. In studying over 100 ISPs, transit providers and content providers for over two years, Labovitz et al. [LIJM+10a] established the changing nature of the Internet topology, migrating from a hierarchy of providers to an increasingly interconnected model. The rise of Internet Exchange Points (IXPs) where customer networks peer directly with content providers has had a profound impact in shaping traffic patterns and interdomain traffic in particular. The length of AS paths has decreased for most *content*. As a consequence, it has become simpler for a stub domain to ensure path disjointness as a larger portion of the end-to-end path is under its control.

The resulting architecture provides many benefits. Balancing is both transport agnostic and allows flow state to be kept to a minimum, requiring policing at the edges only if congestion accountability is required. Additionally, path selection is receiver driven, aligning the stakeholder who can decide when to issue FNE packets with the stakeholder who benefits the most. The key to path selection is the information provided by the sender (point 7 in figure 3.1b), which allows the network to estimate path loss on the same timescale as hosts. By pooling loss information from multiple hosts, the network may additionally be made aware of path failures sooner than hosts using TCP inference.

Chapter 4

Congestion aware traffic engineering

While the previous chapter provides an architecture within which transport and network layers exchange relevant information, it does not define how networks should select outgoing paths. This chapter presents a possible solution for balancing traffic according to expected congestion which works under a wide range of network conditions.

4.1 A model for congestion balancing

Consider LEX-capable traffic, that is explicitly marked as either being retransmitted or non-retransmitted, from a single origin prefix to a single destination prefix, with a number of possible paths and within a single time period. Let N be the number of paths (numbered $1, \ldots, N$). Let T_i be the number of bytes sent down path i for the previous time period. Let R_i be the number of bytes marked as retransmissions down path i for the previous time period. Let $R = \sum_i R_i$ and $T = \sum_i T_i$. While R_i does not strictly represent the number of lost bytes, the ratio of R_i/T_i should be a good approximation of the loss rate within period i. A starting assumption is that it is desirable to equalise the proportion of lost bytes on all paths – that is make R_i/T_i equal for all i – since doing so maximizes social welfare and mimics the behaviour of existing end-host only resource pooling solutions such as MPTCP.

The loss rate $\rho_i = R_i/T_i$ to be balanced is an unknown function of T_i and B_i the bandwidth of the link. It is, however, likely that the loss rate is increasing or at least non-decreasing with T_i. Similarly, the loss rate is decreasing or at least non-increasing with the unknown B_i. Assume then that whatever the true function, for a small region around the current values of T_i and B_i then it is locally linear $\rho_i = k_i T_i / B_i$ where k_i is an unknown constant. Substituting gives

$$B_i/k_i = T_i^2/R_i. \tag{4.1}$$

Now consider the next time period. Use the dash notation for the same quantities in the next time period. In typical time periods $\mathrm{E}[T'] = \mathrm{E}[T]$ since, for example, if the next time period on average had more traffic, the overall amount of traffic would be growing. While this is true in the year-on-year setting, this effect is negligible in the time scale discussed here. Now $\rho_i' = R_i'/T_i' = kT_i'/B_i'$. Choose the T_i' to make all $R_i'/T_i' = C$ (hence all equal) where C is some unknown constant. Therefore $T_i' = CB_i'/k_i'$ but if this is still near the locally linear region then $B_i'/k_i' \simeq B_i/k_i$ and B_i/k_i is determined by (4.1) hence the predicted distribution is $T_i' = CT_i^2/R_i$. Now it is necessary to calculate C by summing over

i. $T = C\sum_i T_i^2/R_i$ and hence $C = T/\sum_i T_i^2/R_i$. This results in a usable loss balancing equation given the available system inputs:

$$T_i' = \frac{TT_i^2}{R_i \sum_j (T_j^2/R_j)}. \tag{4.2}$$

There is, of course, a problem when $R_i = 0$. Given the assumption that the traffic is assigned in inverse proportion to loss rate, a branch with no loss would naturally have all traffic assigned to it. To correct this, R_i is incremented for each path by one MSS. This problem is further explored in subsequent sections.

4.1.1 Understanding the design space

Given local linearity is assumed, large adjustments to the traffic split are a potential cause for concern. Some traffic must also be assigned to each route independently of the loss rate, in order to probe whether the path is available. This leads to a small component of equalisation being used to ascertain that paths are continually used. Finally, in the absence of loss, it is practical to assign traffic according to the current traffic throughput split. This denotes a conservative approach to splitting traffic. Therefore, there are three tendencies which must be accounted for: the loss-equalisation tendency from (4.2), the equalisation tendency to ensure some degree of utilization across all paths, and a conservative tendency to keep the traffic split the same as the throughput split in the previous period. Call the traffic split assigned to path i by each of these schemes $T(E)_i$, $T(C)_i$ and $T(L)_i$ where E, C and L stand for "equal", "conservative" and "loss-driven".

The final distribution of traffic across all links is then:

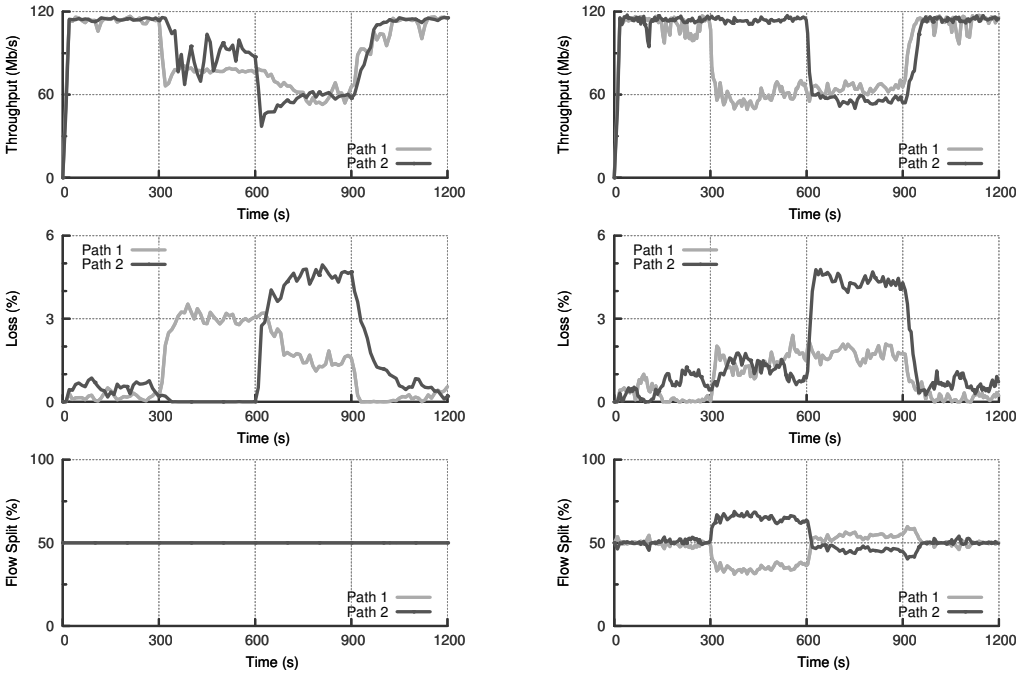
$$T_i' = \beta_E T'(E)_i + \beta_C T'(C)_i + \beta_L T'(L)_i,$$

where the $\beta_\bullet$ are user set parameters in $(0, 1)$ such that $\beta_E + \beta_C + \beta_L = 1$. Now $T'(C)_i = T_i$ and $T'(E)_i = T/N$ where N is the number of interfaces. This gives a first equation for the desired flow split f_i', which represents the probability with which path i will be assigned to a new flow:

$$f_i' = \frac{T_i'}{T} = \beta_E \frac{1}{N} + \beta_C \frac{T_i}{T} + \beta_L \frac{T_i^2}{R_i \sum_j (T_j^2/R_j)}. \tag{4.3}$$

While (4.3) describes a very broad design space, intuition on how each component performs is provided through a simple example.

Consider a network balancing traffic from a source S to a given prefix as shown in figure 4.2. For the current example, only two paths are available with equal bottleneck capacity $L_1 = L_2 = 120$Mbps, For the duration of the experiment, a source balances traffic to C through either link. At $t = 300$s, cross-traffic is generated from server D_1 towards client C through L_1. At $t = 600$s, a greater quantity cross-traffic is generated from server D_2 towards client C through L_2. This topology is later revisited with a greater number of paths in section 4.2, where more details are provided on the simulation parameters used. For the present example, the respective throughput, loss rate and the proportion of flows assigned to each path are shown in figure 4.1 for cases where each traffic balancing component is used in isolation, and a final case in which all three components are used together.



(a) Equalisation.

(b) Conservative.

(c) Loss-driven.

(d) Balanced.

Figure 4.1: Simulation using PREFLEX to balance traffic over two paths. Each simulation demonstrates the dynamics of a single mode, except for (d) which balances between conservative and loss-driven components.

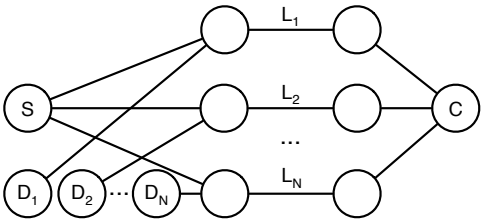


Figure 4.2: Simulation topology.

The effect of equalisation is the first to be considered, given it is necessary to ensure all paths are continually probed. Equalisation alone however leads to an inefficient use of the network if there is a mismatch in path capacity, as made apparent in figure 4.1a as congestion occurs on either link. Such behaviour arises in traditional traffic engineering, which resorts to equalising traffic weighted by local link capacity. Where a bottleneck is remote and distinct however, such behaviour will lead traffic across all links to be roughly bound by the capacity of the slowest path, as can be seen between $t = 300$s and $t = 600$s where throughput over the first path is dragged down by congestion on a second path. Figure 4.1b and 4.1c on the other hand show distinctive behaviour when using a solely conservative or solely loss-driven approach. Predictably for small values of loss the loss-driven approach overreacts and flaps between either path. While the net effect of these oscillations does not result in significant losses, a conservative approach lends itself more naturally to situations where loss is too small to provide a reliable indicator on path quality. Once loss becomes significant however a conservative approach is unable to drive traffic in order to balance loss across both paths. More worryingly, a pure conservative approach demonstrates the wrong dynamics, as highlighted between $t = 250$s and $t = 300$s. Despite being saturated, the balancer continues to push more traffic towards path 1 while the second path remains under-utilized, dropping its throughput further.

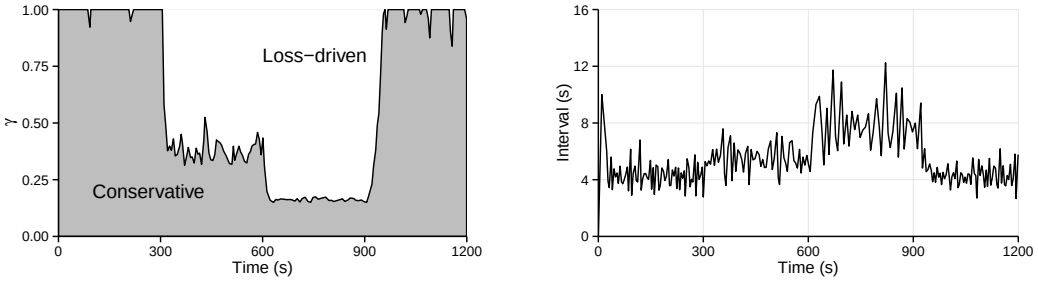
Note that in figure 4.1 the conservative approach obtains higher throughput than its loss-driven counterpart once loss settles in. While this may seem advantageous, in reality this behaviour will be shown to be detrimental to the system as a whole: only by balancing loss can social welfare be optimized [KV05].

4.1.2 Balancing between conservative and loss-driven

Ideally the proposed balancer should adjust between conservative and loss-driven modes for differing regimes of loss. Equalisation is required in some measure as every path must attract some traffic if it is not to fall out of use ([KV05], remark 2). If this were not the case, a path with relatively high loss would never be probed to determine whether congestion has subsided. The remaining two components must be balanced to be able to respond adequately to loss while not being overly sensitive to statistically insignificant fluctuations in loss. Let γ replace both β_C and β_L in (4.3) and adjust between conservative and loss-driven modes:

$$f_i' = \beta_E \frac{1}{N} + (1 - \beta_E) \left(\gamma \frac{T_i}{T} + (1 - \gamma) \frac{T_i^2}{R_i \sum_j (T_j^2 / R_j)} \right). \tag{4.4}$$

As a result, β_E may now vary between $[0, 1]$. A simple but effective value for γ is to define a



(a) γ . (b) τ' .

Figure 4.3: Self-tuning parameters for example in figure 4.1d.

minimum average loss μ_{min} below which there loss is tolerated, and react increasingly as the average loss μ becomes more significant:

$$\gamma = \begin{cases} \frac{\mu_{min}}{\mu}, & \text{if } \mu > \mu_{min} \\ 1, & \text{if } \mu \leq \mu_{min} \end{cases} \tag{4.5}$$

This completes the PREFLEX balancer, as shown in figure 4.1d. The evolution of γ for the example shown in figure 4.1d is shown in figure 4.3(left). The value of μ_{min} was set to 0.005.

4.1.3 Tuning update interval

A further potential issue is how often the flow split should be updated considering the sparseness of loss. Assume that for a given prefix packet loss is measured over a given time period τ. It would be practical to tune this τ per prefix according to the accuracy afforded by the loss estimate. If τ is short then only a very small number of packets will be lost. On the other hand, if τ is long then the control system will be unable to react quickly to changes. τ should therefore be sufficiently small that an accurate measure of loss can be obtained. As a result, it is useful to have a rough estimate of the measurement period required to ensure accurate loss estimates. Since this time period of measurement is per prefix, this must to some extent reflect the importance a given measurement to the system as a whole. For example, the overall system should not be held up by a single route with insufficient traffic for an accurate measurement to be obtained. This will be achieved with the concept of a weighted coefficient of variation.

Let t_i be the number of packets transmitted down path i in the time period τ and let l_i be the number of packets which were lost in this time period. To prevent the aforementioned divide by zero issue, $l_i = 1$MSS if no packets are lost.

Let p_i be the probability that a given packet is lost on path i and assume that packet loss is a Bernoulli process. An unbiased estimate of p_i is $\hat{p_i} = l_i/t_i$. It is important to what follows that $\hat{p_i}$ is only an estimate of p_i by "chance" more or fewer packets may have been lost. Given packet loss is Bernoulli then l_i has a binomial distribution and its variance σ^2 is given by $t_i p(1-p)$. The coefficient of variation (CV), is a dimensionless measure given by the standard deviation over the mean $c_v = \sigma/\mu$. Keeping the coefficient of variation within some bound δ is a measure of the amount by which an estimate is likely to vary from the true mean.

From trivial definitions of σ and μ, for the number of lost packets on a given prefix i the estimated CV is $\hat{c_v}(i) = \sqrt{t_i \hat{p_i}(1-\hat{p_i})}/t_i\hat{p_i}$. Let r_i be the rate of packet arrival per unit time on i giving $t_i = r_i\tau$. Define W the CV weighted by transmitted packets over the prefix as $W = \sum_i t_i/t\hat{c_v}(i)$ where $t = \sum_i t_i$ and this expands as

$$W = \sum_i \frac{t_i}{t}\sqrt{\frac{(1-\hat{p_i})}{r_i\tau}}.$$

The "accuracy" of the measurement of p_i is determined by the accuracy of l_i and hence, for the prefix as a whole by the CV W. The aim now is to pick the time period for the next measurement τ' such that $W \leq \delta$ for some δ. Assuming that the loss rates and traffic rates will be the same in the next time period will give a good indication of how to set τ'. Therefore for the next time period

$$\delta \geq W = \frac{1}{\sqrt{\tau'}}\sum_i \frac{t_i}{t}\sqrt{\frac{(1-\hat{p_i})}{r_i}}.$$

This gives an estimated minimum time period to set for the next time period. In order to get weighted CV of packet loss (and hence loss rate) equal to or below δ the time period τ' is bounded by

$$\tau' \geq \tau\left(\frac{1}{\delta t}\sum_i \sqrt{t_i - l_i}\right)^2.$$

This equation gives the smallest value to set the time period of measurement to in order that the weighted coefficient of variation of the loss measurement is a given δ. An upper bound for this value should be set by operators to ensure that the system periodically updates flow splits when operating under extremely low loss rates.

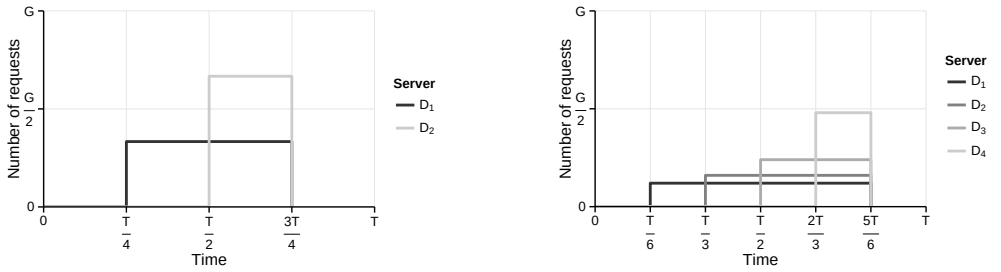
While a number of simplifying assumptions have been made, such as modelling loss as Bernoulli, the time scale choice is not a critical system parameter so long as it provides "good enough" estimates of loss. The evolution of τ' for the example in figure 4.1d is plotted in figure 4.3. As throughput decreases, the time between updates is inflated to adjust to the lower occurrence of loss events.

4.2 Performance Analysis

This section evaluates PREFLEX through simulation using ns-3 [ns3]. Evaluating traditional traffic engineering methods typically involves abstracting traffic as flow aggregates, making large scale network performance analysis tractable. PREFLEX however balances traffic using loss rather than load, and focuses on improving end-user metrics as opposed to minimizing maximum link load. As such, evaluating the proposed congestion balancer requires simulating end-to-end behaviour of traffic.

4.2.1 Methodology

Experimental validation is performed using the network topology previously displayed in figure 4.2. The topology links a client domain C to a server domain S through N paths with bottlenecks L_i, and total bandwidth $B = \sum L_i$. Client C generates G simultaneous HTTP-like requests (or "GETs") from S according to a specified distribution, described at the end of this section. As traffic flows from S to C, the router within S is responsible for balancing traffic over all available paths. This topology is chosen since it provides multiple independent bottlenecks over which domain S must balance elastic TCP traffic



(a) $N = 2$. (b) $N = 4$.

Figure 4.4: Number of requests from C to cross traffic servers D_i for different values of N

without any explicit knowledge of the conditions over links L_i. Under such conditions traditional traffic engineering methods may perform poorly since they resort to local information only.

Across simulations, as the number of paths increases, total bandwidth B and the number of simultaneous requests G is fixed, providing insight into how efficiently PREFLEX balances traffic as the granularity with which it can split traffic becomes coarser.

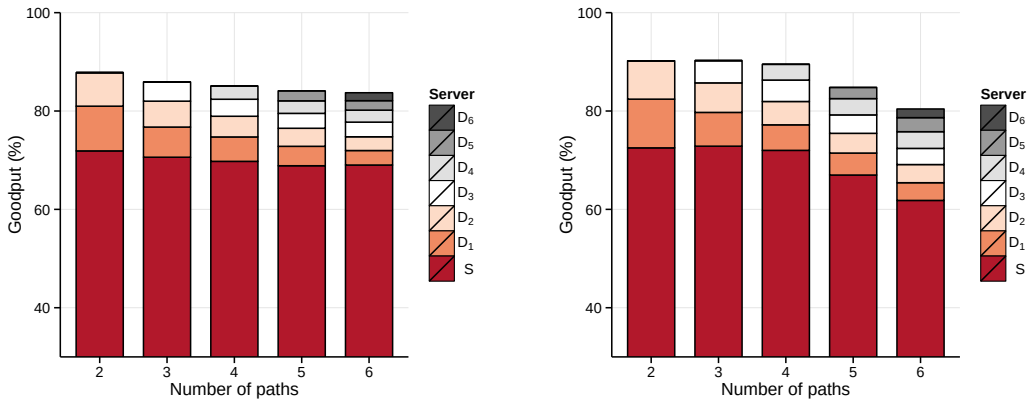
In order to evaluate how PREFLEX shifts traffic in response to loss, additional "dummy" servers D_i are connected to C through bottleneck link L_i. The simulation runs for time T and is partitioned into $N + 2$ intervals starting on s_i, in which s_0 and s_{N+1} have no traffic to D_i. Starting at time s_i, client C generates g_i requests to D_i according to the same distribution as used to server S. All requests to D_i end at time s_{N+1}. Equation (4.6) sets the start time s_i for requests to D_i as a function of total simulation time T and number of paths N. Likewise, equation (4.7) sets the number of simultaneous requests g_i to D_i as a function of G, the total number of requests to S, and N.

$$s_i = T\frac{i}{N+2} \tag{4.6}$$

$$\theta_i = \frac{\frac{1}{N+1-i}}{\sum \frac{1}{N+1-i}}, g_i = G\theta_i. \tag{4.7}$$

Figure 4.4 illustrates the number of simultaneous gets from C to D_i for $N = 2$ (used in the example shown in figure 4.1) and $N = 4$. Generating cross-traffic in this manner serves two purposes. Firstly, $\sum g_i = G$, so independently of the number of concurrent paths, the maximum load in the system is $2G$. However, as the number of paths increases, the fluctuation in load for each path becomes smaller, stressing the sensitivity with which PREFLEX must balance traffic. Secondly, the number of requests for each D_i over time is the same. Over timescale T, equalisation appears to be an acceptable strategy but will however be shown to fail to make efficient use of available capacity. This is a fundamental limitation of offline traffic engineering, which is calculated over long time scales and is unable to adapt as traffic routinely shifts.

The settings used for all simulations, including those previously shown in figure 4.1, are as follows. Total simulation time T is set to 1200 seconds, while total bandwidth B is fixed at 240Mbps. The number of requests G sent from C to S is set to 240. Upon completing, a request is respawned after



(a) Equalisation.

(b) PREFLEX.

Figure 4.5: Goodput relative to B achieved by each server over equal capacity links according to different balancing strategies.

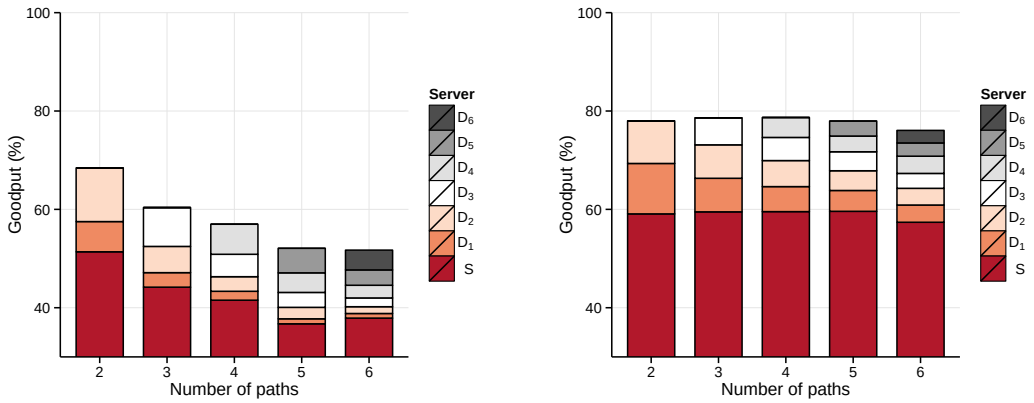
an idle period following an exponential distribution with a 15s mean. Transfer size follows a Weibull distribution with an average value of 2MB. While artificial, these values attempt to represent traffic to a single prefix with a file size that mimics the small but bursty nature of web traffic, which does not lend itself to being balanced by the end-host, since flows are too short to individually collect reliable information on network conditions. PREFLEX is configured with $\beta_E = 0.05$, $\mu_{min} = 0.01/N$ and $\delta = 0.005$.

4.2.2 Varying bottleneck distribution

For the remainder of this section, congestion balancing using PREFLEX will be directly compared to equalisation, which mimics existing traffic engineering techniques based on hashing flow tuples for path assignment. A useful reference point in interpreting results is to examine the case where all bottlenecks share the same bandwidth, $L_i = B/N$. Under such conditions, figure 4.5 shows the goodput, calculated as the total data transferred to client C by flows completed within T, as a proportion of total link bandwidth. Goodput is selected to compare between balancing mechanisms since it is a transport metric and therefore a more accurate representation of the throughput attained by applications. The bulk of goodput originates from server S, which is the only multi-homed domain. If traffic is correctly balanced, servers D_{1-N} should generate the same amount of goodput. While both equalisation and congestion balancing saturate most available bandwidth, the former leads to disproportionate distribution of goodput amongst competing traffic. When loss is not equalised over all paths, the amount of goodput achieved by servers D_i differs despite demand being similar.

For the case where all bottlenecks are equal, equalisation can be seen as the optimal static TE solution, yet both approaches bear similar performance. With no knowledge of topology, link bandwidth or expected traffic matrices, PREFLEX is able to adequately mimic the performance of the static TE solution for the case where such an approach is best suited.

Where bottleneck bandwidth is unequal however equalisation is shown to be severely lacking. The



(a) Equalisation.

(b) PREFLEX.

Figure 4.6: Goodput relative to B achieved by each server over unequal capacity links according to different balancing strategies.

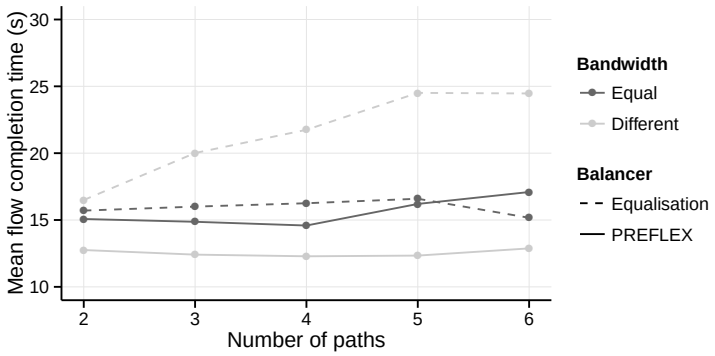


Figure 4.7: Mean flow completion time for equal and differing bottleneck links.

effect of differing bottlenecks is investigated by repeating previous simulations with the same total bandwidth B, but with L_i set proportionally to B in a similar manner to (4.7), that is $L_i = \theta_i B$. The ensuing results, shown in figure 4.6, highlight two significant shortcomings of equalisation which PREFLEX overcomes. Firstly, goodput for S drops as N increases. Unable to realize it is overloading a path, equalisation is reduced to sending traffic over each link at approximately the same rate as the most congested link. In contrast, PREFLEX detects congestion and adapts accordingly. Secondly, the incorrect distribution of traffic due to equalisation in S distorts the goodput of competing traffic. While in PREFLEX goodput from D_{1-N} is perfectly balanced, with equalisation traffic crossing the most congested links are directly affected by another domain's inability to distribute traffic appropriately. It may seem unfair to judge equalisation for cases where there is a mismatch in link capacity. However, such a mismatch between link weight and path capacity arises regularly as operators adjust traffic engineering according to local conditions, with little thought spared for the impact this may have further downstream.

This impact is in turn perceived by users, who experience longer flow completion times, as shown in figure 4.7. In the equal bandwidth case the flow completion time is similar for both balancers. Where

bandwidth differs however, balancing by congestion outperforms equalisation and maintains a stable performance even for all six paths. This shows that the algorithm scales well as the number of available paths increases.

4.3 Conclusions

This chapter introduced congestion balancing using PREFLEX. PREFLEX has been implemented and evaluated in ns-3 for dynamic traffic scenarios where it balances traffic using different strategies which are weighted according to the network conditions it detects. In conditions where loss is deemed significant, PREFLEX balances congestion between paths. In the absence of sustained loss PREFLEX assigns traffic based on current throughput. By balancing between these strategies PREFLEX can operate in a variety of dynamic traffic settings and has been shown to perform as well as the ideal static traffic assignment bandwidths are equal. Where bandwidth asymmetry arises, PREFLEX successfully balances loss with no significant degradation of performance as both the number of paths and inherent complexity of balancing increases.

Chapter 5

A longitudinal analysis of transit traffic

By readjusting traffic according to end-to-end metrics, PREFLEX is unique in proposing congestion, rather than just load, as an essential metric for traffic engineering. In the previous chapter necessarily artificial end-to-end behaviour was used in order to gain insight into how PREFLEX works. Understanding the extent to which PREFLEX can benefit end-users and networks in practice however requires a deeper understanding of the inherent characteristics of Internet traffic at large.

This chapter provides a longitudinal analysis of the characteristics of end-to-end Internet traffic, describing the shifting trends in interdomain traffic as viewed from WIDE, a Japanese academic provider. Over time, this vantage point is subject to upgrades, changes in routing policy and congestion events, all of which can hinder the interpretation of data. These limitations are overcome by looking further afield, searching for clues within shifts in the geographical and topological make-up of inbound and outbound traffic and how these trends relate to end-to-end performance.

5.1 Related work

Despite their inherent value, longitudinal studies of Internet phenomena are rare. Over its short lifespan the Internet has been shaped as much by technological change as by political and commercial realities. This dynamic nature does not lend itself to observational studies where data must be collected and curated over long periods of time, and has resulted in a scarcity of relevant datasets. What few exceptions exist often stem from collaborative research efforts, such as CAIDA [CAI] or Oregon Routeviews [rou]. The usefulness of these datasets however can be severely affected by the need for data privacy. The dissemination of interdomain routing information, where no such requirement exists, has assisted in a wealth of research on wide ranging topics, from quantifying path diversity [OZPZ09] to locating Internet bottlenecks [HLM+04]. In contrast, longitudinal datasets relating to passive measurements have nurtured a much smaller community of researchers often focusing on characterizing traffic [FBAF10]. Stripped of the locality contained within IP addresses however, researchers are left unable to relate these findings to a wider context. Instead, cross-sectional studies characterizing traffic aggregated by location are frequently conducted under different contexts [AMSU11], but lack the temporal perspective only longitudinal studies can afford. Efforts to characterize the spatial properties of traffic over time [DD11, LIJM+10b, CFEK08] have defined the changing of Internet topology and traffic alike but fall

short of relating such shifts with their impact on relevant metrics such as loss or delay.

This chapter builds on a wealth of prior work on understanding Internet traffic and serves as a reappraisal of significant past contributions. Flow characteristics and TCP behaviour at large are subject to frequent reassessment [ZBPS02]. Of particular relevance to the current work are passive studies which delve into the inner mechanisms of TCP. In [JID+04], Jaiswal et al. infer the sender's congestion window by identifying the congestion control variant from the behaviour observed during loss recovery. The use of separate state machines for each variant however proves unscalable given the many flavours of TCP congestion control which have since been deployed. In [LH06], Lan et al. analyse flows according to size, duration, rate and burstiness and characterise the observed correlations for heavy-hitters specifically, uncovering evidence of increased application influence on flow rates and burstiness and consequently suggest treating flow size and duration as independent dimensions.

One central aspect to the analysis of TCP behaviour is the estimation of RTT from packet capture data. In addition to SYN-based methods, Shakkotai et al. [SSB+04] evaluate further techniques to estimate the RTT of a unidirectional flow. The *rate change* method establishes a relation between the RTT and the increase in sending rate, assuming linear window increases during congestion avoidance. Unfortunately, this assumption no longer holds, both due to the proliferation of less conservative congestion control algorithms such as CUBIC [HRX08], and due to application-driven flow control. An alternative is the use of frequency-domain techniques [VLL05, LF05, QGM+09], which are a natural fit given the self-clocking nature of TCP. However, a common difficulty with the application of spectral analysis is extracting the fundamental frequency which corresponds to the RTT in the presence of noise. In applying the Fourier transform to inter-packet arrival times, for example, Qian et al. [QGM+09] note that less than half of all flows have distinguishable *flow clocks*; likewise, the Fast Fourier Transform (FFT)-based RTT recovery was found to be unreliable even after pre-processing available data to enhance inherent periodicities.

Finally, it is important to elucidate what changes in traffic properties are intrinsic to TCP and data transfer, and which ones arise from large-scale changes in the AS-level topology of the Internet. In the decade since publication of [ZBPS02], the Internet has undergone significant changes, shifting from a broadly hierarchical form to a flatter, more interconnected structure [LIJM+10a, ACF+12]. Given the longitudinal nature of this chapter and its focus on interdomain traffic in particular, the insights provided by these studies on the macroscopic effects of content consolidation are discernible within the studied dataset, and as such are a source of validation for many of the observations herein.

5.2 Dataset

This section provides an overview of the datasets used in this work and some of the data processing required before approaching the longitudinal study of Internet traffic rate limiting. The dataset used is composed from the original, un-anonymised traffic traces from the Measurement and Analysis of the WIDE Internet (MAWI) dataset [CMK00], a set of daily packet captures from the WIDE backbone network which provides connectivity to universities and research institutes in Japan. Traffic is collected daily for 15 minutes starting at 14:00 JST. Although this dataset extends back largely uninterrupted

from late 2001, the present work focuses on just over five years of data following a network upgrade to the monitored link on October 2006. The monitored link carries mostly trans-Pacific commodity traffic between WIDE customers and non-Japanese commercial networks. Traffic towards WIDE is referred to as *inbound* traffic, whereas traffic originating from within WIDE is referred to as *outbound* traffic.

		TCP DATA	TRAFFIC (TB)		UNIQUE ($\times 10^3$)	
YEAR	DAYS	FLOWS ($\times 10^3$)	IN	OUT	AS	PREFIXES
2006	91	20.52	0.43	0.45	10.90	56.86
2007	350	102.56	2.11	2.49	17.21	113.79
2008	358	112.26	2.43	2.10	24.74	156.54
2009	364	113.97	2.48	2.53	19.71	143.87
2010	365	113.70	2.58	3.43	20.38	148.03
2011	358	114.74	3.44	5.14	19.99	140.56
TOTAL	**1886**	**5777.55**	**13.50**	**16.14**	**34.12**	**341.22**

Table 5.1: Overview of traced MAWI dataset.

A preliminary overview of the dataset used is provided in table 5.1. In total, 5.7 billion flows containing data are traced over five largely uninterrupted years; this represents approximately 30 terabytes of TCP traffic. For the purposes of this work, most analysis will focus on inbound traffic, 60% to 80% of which originates from port 80, referring only to analysis of outbound traffic when contextualizing findings. Given the sender side plays a critical role in shaping traffic, analysing traffic for which the source is restricted to a small set of networks within Japan is of limited use in accurately depicting traffic trends at large. Hosts within Japan are instead fixed as traffic sinks, thus sharing a similar perspective on inbound traffic as many other similarly sized networks.

5.2.1 Tracing TCP Metrics

All TCP flows are reassembled and analysed for each daily trace. In addition to the five tuple used to define each connection, two additional restrictions are imposed: a contiguous sequence number space and a three minute timeout. These restrictions are helpful to deal with port reuse and unterminated flows respectively. Although the total number of TCP flows increased dramatically in 2011, the number of flows for which data payload was observed has remained stable, averaging over 100 million data flows traced per year.

There is much prior work with regards to reconstructing TCP flow from passive measurements and using this information to understand the end-to-end properties of traffic [MMC00, JID$^+$07, RKS07, SSB$^+$04]. However, the MAWI traces impose two constraints which require careful consideration, and ultimately led to the use of a custom TCP tracer. The first is the proportion of bidirectional flows, where both forward and reverse path are seen. In the dataset used this fluctuates between 40% and 60% over five years. Most available TCP tracers either ignore or are inadequate at processing unidirectional flows. The second is the short duration of each individual trace file. At only 15 minutes of line-rate data capture

per day, it is wasteful to ignore flows which are not complete. Although the number of flows for which a SYN and FIN in either direction is observed has remained consistently high until late 2011, these flows are normally *mice*, i.e. flows that tend to be brief and which carry little traffic individually. In contrast, most *elephants* (flows that carry significant traffic individually) have durations that exceed that of each trace file.

Loss is inferred by accounting for *retransmissions* in the upstream data and *out-of-order packets* in downstream data; for the remainder of the paper the term *end-to-end loss* will refer to the sum of out-of-order and retransmitted data bytes over the total data bytes in a given direction. Anecdotally, this was found to be an adequate indicator of loss — with the exception of *hanging* TCP connections. In such cases where connectivity is lost, a host will proceed to retransmit packets while performing an exponential back-off. Although this results in negligible overall traffic, it can significantly skew the inferred loss ratio for uncommon destinations for which little traffic exists. To account for these cases, a 3-second timeout on retransmissions was imposed, after which the congestion feedback loop is considered to be broken.

Each daily trace in the dataset is processed from a packet level capture into a collection of flow level statistics, providing insight into the end-to-end characteristics of traffic. However, since a core objective of this work is to augment this time-based information with data describing the endpoints of each flow, aggregating by location is also required.

5.2.2 Aggregating by Location

Location information is added by mapping the original source and destination IP addresses to its geographical and topological counterpoints. The routeviews archives [rou] are used to reconstruct the mapping between each IP and both AS and network prefix; bi-hourly dumps of BGP Routing Information Bases (RIBs) are available in the WIDE archives since mid 2003. A daily RIB is reconstructed based on the views provided by contributing ASes, in particular IIJ and APNIC. Since there is no record of local policy, exact routes are not disclosed and as such there is no prior knowledge of the route taken by packets; this however does not hinder the ability to consistently map IPs to ASes. While discrepancies in AS destinations exist between different routeviews contributors, this happens almost exclusively on prefixes for which no actual traffic is seen.

Mapping IP to country is done through the use of GeoLite [max12], a commercial geolocation database. While the accuracy of this solution is often disputed, locating traffic at a fine granularity is not a pressing concern. Most geographic emphasis will be placed on capturing macroscopic shifts in time at a national level, for which Geolite proves adequate. The archive for geolocation data only extends to 2009, before which the earliest match must be used. Additionally, the administrative mapping up until mid 2009 for a destination or source AS is verified to have remained the same in the relevant Routing Information Registrar (RIR) archives in order for a flow to be assigned a geographical location. After associating flows to country, region, AS and network prefix for both source and destination IPs, flow statistics are aggregated over each location identifier. This generates a daily collection of location identifiers and associated flow properties, from which the geographic and topological properties of the

dataset can be sketched over time.

5.3 RTT estimation

Building on prior work presented in section 5.1, this section proposes an algorithm that scalably recovers the RTT from one-directional traffic traces. Although RTT estimation is a difficult problem, simplifying assumptions can be made. For the MAWI dataset most RTTs are relatively large, with the closest neighbouring country, South Korea, roughly 40ms away. By only processing bidirectional traffic from Japan, the expected RTT range can be reduced for all other traffic. The recovery mechanism then enhances the natural periodicity of traces and scalably constructs flights associated with specific application and protocol behaviour. In the following the mechanisms required by these two goals are described. In normal operation, many TCP operations involve request-response cycles between two endpoints in which the RTT T provides a natural *clock*. Hence, the most natural way to estimate RTT from TCP traces is to correlate requests and responses exchanged in both directions. If only one direction of data is observed however, T cannot be directly observed. Instead, it must be estimated from the way in which TCP packets cluster in time due to the batching of request-response operations.

The TCP cwnd determines the number of unacknowledged bytes that a TCP flow may maintain at any point in time. This can be referred to as *bytes in flight* because they are in transit between the sender S and the receiver R; an equivalent definition applies for the number of *packets in flight*. Once S has transmitted cwnd data bytes, it will refrain from transmitting more until either some bytes are acknowledged by R or cwnd is increased by the sender. In the absence of losses, neither of these events can happen until a TCP ACK is received; this immediately reduces the number of unacknowledged bytes, but may also lead to a significant cwnd increase (during e.g. *slow start*). In the presence of losses, however, bytes can be re-sent if a packet is timed out and considered lost; in this case, the number of unacknowledged bytes is reduced.

The main difficulty associated with one-sided TCP flow reconstruction is as follows. Let $t_1, t_2, \ldots$ be a set of times at which packets $p_1, p_2, \ldots$ were observed at S en route to R. Suppose that a packet p_j of size b is observed at time t_j. In addition, suppose that approximately one RTT T later, the sender S receives an ACK a_j from R for the b bytes of p_j. At this point, the TCP stack in S will decrease the number of unacknowledged bytes by b, thus opening the possibility for sending additional traffic to R. This can lead to another packet p_k to be transmitted; let this packet be observed at time t_k as it is sent towards R. Assuming that processing delay is insignificant, the RTT experienced by p_j can be approximated as $T \approx t_k - t_j$. Now consider what happens if packets are only observed in the $S \rightarrow R$ direction. Under such conditions, it is not possible to ascertain whether p_k was sent explicitly as a result of S receiving the unobserved ACK a_j, or whether it was sent as a result of an ACK a_i associated with a previous packet p_i rather than with p_j. If, however, a packet p_l is eventually observed that did result from the reception of a_j, the RTT can be estimated as $T \approx t_l - t_j$ with $t_l > t_k$. Following this same reasoning, approximately one RTT later a packet p_m will be observed for which $2T \approx t_m - t_j$; this can potentially continue for as long S has data to send and R continues sending ACKs. This is the underlying reason that RTT-related periodic regularities arise when considering the timestamps of observed packets

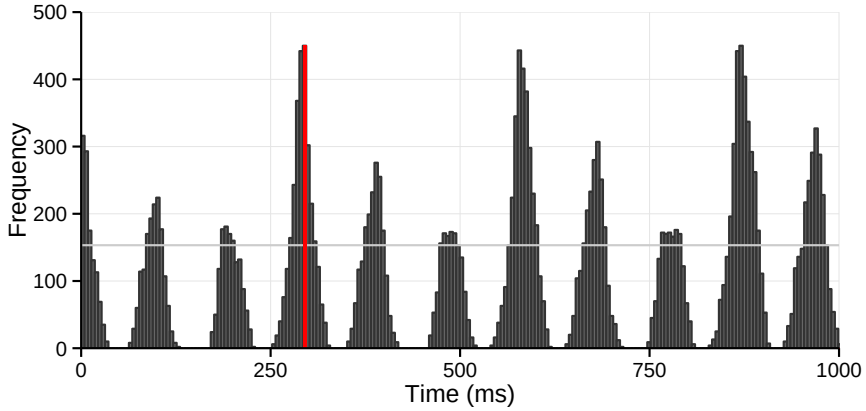


Figure 5.1: $H(t)$ for example flow. The horizontal line delimits $\overline{H}$ while the highlighted bin denotes the bidirectional RTTs estimate.

[QGM+09].

The reasoning above is at the heart of the proposed algorithm to improve RTT recovery by enhancing packet stream periodicity. Assume that a packet p_j is observed at time t_j. Considering the set $\mathcal{T}_j$ of all values of $\Delta t = t_k - t_j$ for every $k > j$, it is apparent that it will include estimates not only for the RTT T, but also for all its multiples $2T, 3T, \ldots$ If $t_l - t_k \approx T$ and $t_k - t_j \approx T$ then it follows that $t_l - t_j = 2T$, and this value will also be included in $\mathcal{T}_j$.

By maintaining a set $\mathcal{T}_j$ for every packet p_j observed, at least some of its values will correspond to estimations of multiples of the RTT. It then follows that by creating a set $\mathcal{T}$ that includes values calculated starting from every packet p_j so that $\mathcal{T} = \cup_j \mathcal{T}_j$, numerous estimates for $2T, 3T, \ldots$ will also be included. Hence, the probability density function $H(t)$ of the values in $\mathcal{T}$ should show peaks around multiples of the RTT (see Figure 5.1).

The algorithmic recovery of T from $H(t)$ presents additional challenges. In particular, $H(t)$ may include a large number of RTT multiples, and a peak will be found for all of them. Crucially, all these peaks may be of comparable magnitude, complicating the task of selecting a single peak. Moreover, these peaks need not be very pronounced, with histogram bins in close proximity of the peaks have very similar values as the peak itself. As such, taking RTT candidates directly from $H(t)$ may result in a large set of similarly-valued bins situated around a peaks at multiples of the RTT.

Three recovery algorithms for T are attempted. First, as a baseline, the highest peak in $H(t)$ is selected as a candidate for T. In addition, expanding upon the work of Qian et al. [QGM+09] a frequency-domain representation of $H(t)$ is used to identify T. This is done by selecting the highest peak of $|\hat{H}(\omega)|^2$, the *energy spectral density* of $H(t)$ (i.e. the norm squared of the Fourier transform of $H(t)$). Finally, a custom utility-based technique that operates directly on $H(t)$ is proposed which achieves superior performance to both of the aforementioned methods.

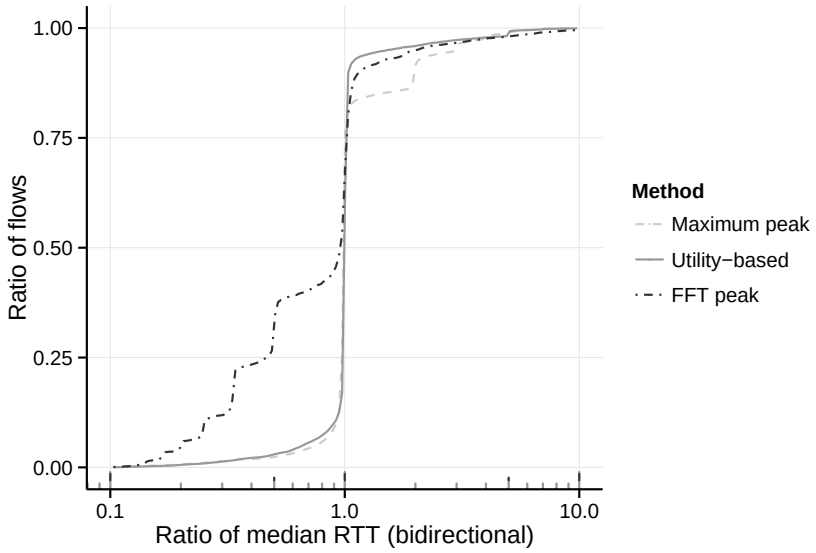


Figure 5.2: Accuracy of RTT estimator when compared to the median value of bidirectional estimate.

5.3.1 Utility-Based RTT Recovery

This method relies not on the identification of periodicities, but on explicitly matching experimentally found signatures. To this end, we consider the peaks of $H(t)$, which are then considered RTT candidates. However, trivial discriminators (such as simply selecting the highest peak) are not reliable. In this case, it was found experimentally that repeatable peaks and troughs also occur at multiples and sub-multiples of T, with the most important ones being $\frac{T}{3}$, $\frac{T}{2}$, T and $2T$. We design this detection algorithm around the idea that a given pattern of peaks and troughs can identify T.

If we define $\overline{H}$ as the mean height of $H(t)$, we can define a per-peak utility function $p(t)$ so that

$$p(t) = 1.0 - \exp\left(-2.0\left(\frac{H(t)}{\overline{H}}\right)\right).$$

This function has several advantageous properties: it is 0 if $H(t)$ is zero, 1 if $H(t)$ is infinite, and 0.5 if $H(t) = \overline{H}$. In other words it is a measure of the *peakiness* of the data, with $p = 1$ identifying an infinitely high peak, $p = 0$ identifying an empty histogram bin (trough), and $p = \frac{1}{2}$ implying that $H(t)$ is of exactly average height at that point. We can then score each candidate using the following utility function:

$$P(t) = 1.5p(t) + p(2t) - p\left(\frac{t}{2}\right) - p\left(\frac{t}{3}\right).$$

That is, the candidate RTT t scores highly if it is itself a peak, if it has a peak at a multiple $2t$, and if it also manifests troughs at sub multiples $\frac{T}{2}$ and $\frac{T}{3}$. The factor of 1.5 was added after observations showed that the peak at T was the most important factor in determining whether a candidate was the true RTT. Similarly, additional multiples and sub multiples were excluded as they showed very limited discriminating power experimentally.

	PEAK (%)		UTILITY (%)	
SAMPLE DATA	BELOW	ABOVE	BELOW	ABOVE
RECEIVER SIDE				
$flow <$ 10MB	4.31	9.13	4.58	6.35
$flow >$ 10MB	6.72	6.43	4.97	5.33
SENDER SIDE				
$flow <$ 10MB	2.94	8.37	3.29	4.80
$flow >$ 10MB	6.41	9.06	5.40	11.06

Table 5.2: Performance of peak-based and utility-based RTT recovery algorithms, showing proportion of flows for each sample dataset with estimates below and above bidirectional estimate.

5.3.2 Comparing recovery algorithms

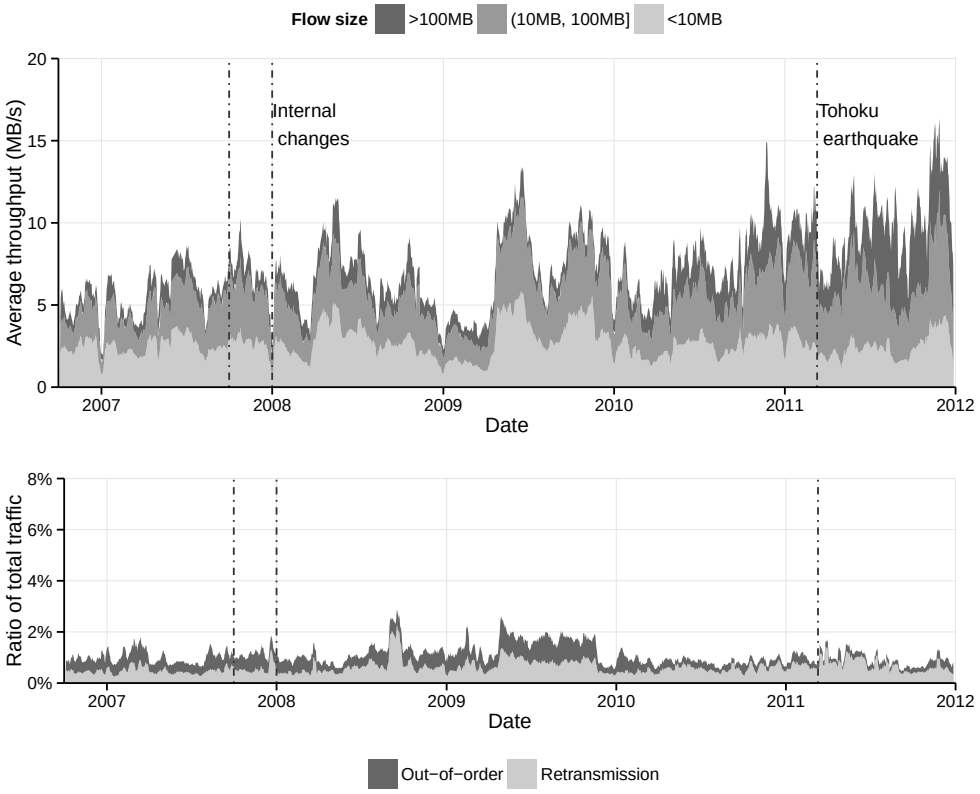
As described in section 5.3, $H(t)$ is calculated in such a way that RTT periodicity is amplified. This means that FFT-based techniques could potentially perform better on $H(t)$ than on the packet stream with no pre-processing. However, this is complicated not only because $H(t)$ contains periodicities at multiples of T, but also discontinuities that generate harmonics at frequency multiples of the RTT fundamental. Hence, although the FFT $|\hat{H}(\omega)|$ of $H(t)$ is much cleaner than that of the packet inter arrival time series on its own, its maximum peak rarely coincides exactly with the RTT clock (this corroborates reports by Qian et al. [QGM+09]). Thus, applying the FFT leads to another *peak detection problem* in which the RTT fundamental needs to be extricated from its harmonics and sub-harmonics. The trivial solution to this problem, the application of a bandpass filter around the RTT frequency, is of course infeasible because the bandwidth and centring of such a filter depend on the RTT which is itself unknown. The utility-based algorithm described in Section 5.3.1 can hence be applied in either the time domain or the frequency domain; we chose to do it on the former on the interest of expediency and lower computational cost.

The performance of the analysed RTT recovery mechanisms is presented in Table 5.2, that shows the percentage of total flows below and above the RTT range given by the bidirectional estimates. We separate things for *inbound* traffic (where we are positioned at the receiver side) and *outbound* traffic (where we are positioned at the sender side). The utility-based algorithm is particularly useful to address RTT underestimation for flows over 10MB in size, which is our main objective since precisely that kind of estimation error would interfere with our ability to correctly decouple application behaviour from RTT-scale dynamics.

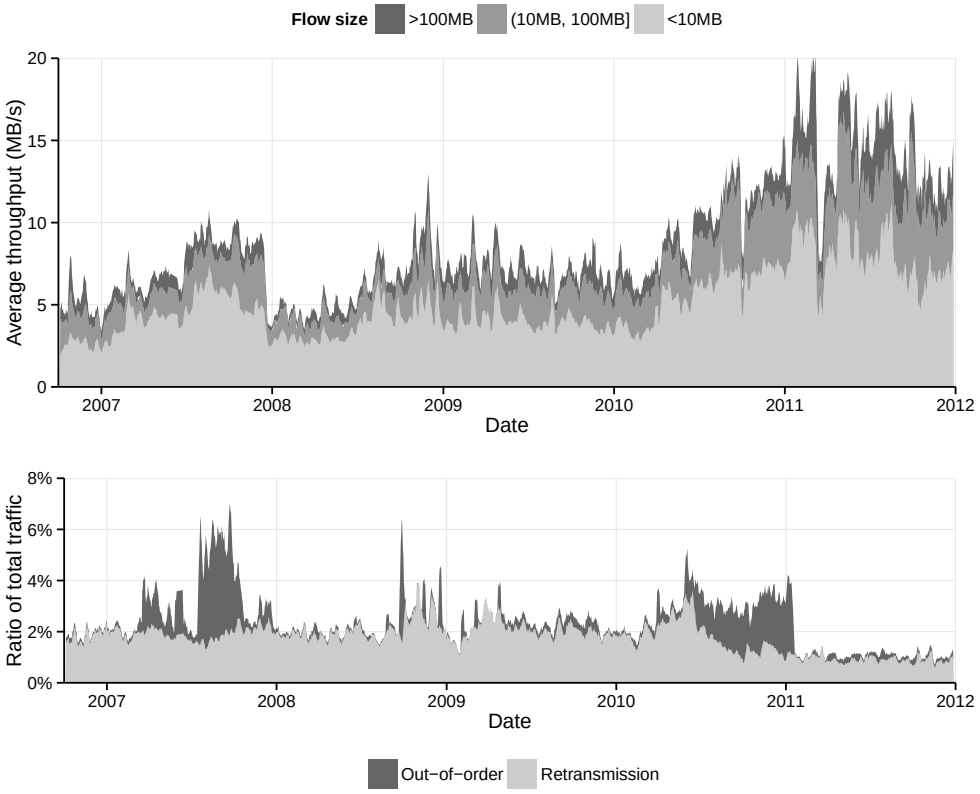
5.4 Macroscopic traffic trends

Over a five year period, changes in routing and application popularity have continually redefined the nature of traffic under observation. This section provides a macroscopic view of these shifting trends. Figure 5.3 displays the average throughput and loss ratio for traffic in either direction, calculated for TCP traffic only, smoothed on a weekly basis.

For inbound traffic, shown in figure 5.3a, two routing changes internal to WIDE had significant



(a) Inbound traffic.



(b) Outbound traffic.

Figure 5.3: Longitudinal evolution of average throughput and loss for the MAWI dataset.

impact on overall traffic, and are consequently highlighted. The first, performed towards the end of 2008, diverted most of the inbound traffic from *national* sources away from the monitored transit link, resulting in a reduction of traffic. This event was preceded by increased congestion downstream from the monitoring point. The second, in early 2009, saw a significant increase in *regional* traffic from Asian neighbours, and was reverted approximately six months later. During this period aggregate end-to-end loss rates increased as a result. While this is mostly due to the higher proportion of upstream congestion for traffic from Taiwan and China in particular, most traffic was adversely affected by the increased utilisation, suggesting that the transit link itself may have been a bottleneck during this period. Finally, the impact of the Tohuku earthquake resulted in a noticeable break in demand coinciding with the start of the Japanese fiscal year in April, in which traffic traditionally ramps up.

For reference, relevant metrics for outbound traffic are also shown in figure 5.3b. Compared to inbound traffic, outbound traffic is subject to higher rates of out-of-sequence packets. The proportion of out-of-order packets in particular is large for extended periods of time at the end of both 2007 and 2011, ending abruptly. This either suggests that the internal network was congested and eventually upgraded, or that load-balancing mechanisms leading to increased network reordering were employed and then decommissioned. The higher rate of retransmissions on the other hand is largely related to the geographic differences between traffic sources and traffic sinks in the MAWI dataset.

5.4.1 Geographic distribution

This skew in the location of end points is apparent in table 5.3, which highlights the geographic distribution of both inbound and outbound traffic for the observed time period.

The majority of traffic flows to and from the United States, which has increased its share of bandwidth in either direction over the past five years. The proportion of traffic flowing from the United States is particularly high, accounting for almost 70% of inbound traffic in 2011. While this may foreshadow an increased concentration of traffic from the United States, it should primarily be viewed as a reflection of routing policy, with regional traffic being diverted to alternate routes as Japan became increasingly interconnected to its neighbours. Of particular importance is the routing change at the end of 2008, which resulted in a sharp drop in inbound traffic from within Japan, as highlighted in figure 5.3a. This event had a profound influence in shaping not only the distribution of traffic, but also delay as shall be observed in section 5.4.3.

Further geographic shifts in the inbound direction are apparent when breaking down US traffic by state. The proportion of traffic originating from California has decreased over time, dropping from 55% of total US traffic in 2007 to only 35% in 2011. In its place, a larger set of states have emerged as content providers, with New Jersey, Florida and Virginia contributing over a quarter of all traffic originating within the US by 2011.

In the outbound direction, the geographic distribution of traffic is less skewed, with a greater proportion of traffic flowing towards Japan and China in particular. Traffic to the Republic of Korea progressively increases from 2010 onwards due to successive routing changes. Combined with the drop in traffic towards China, this accounts for much of the drop in aggregate loss rates since 2010, as observable

Country	Outbound traffic (%)					Inbound traffic (%)				
	2007	2008	2009	2010	2011	2007	2008	2009	2010	2011
United States	27.3	31.3	29.3	36.4	35.7	45.7	41.5	53.3	65.1	67.1
California	39.0	61.8	63.5	53.8	50.6	55.7	47.9	46.7	24.9	34.9
Texas	5.8	4.3	4.1	2.4	13.9	7.0	12.0	5.8	7.1	5.6
Colorado	1.9	1.2	0.6	8.5	2.8	4.9	6.0	5.9	9.7	5.8
Virginia	1.9	1.0	0.8	0.4	0.6	1.2	3.0	14.1	13.1	8.3
Washington	4.0	2.9	3.5	6.1	6.6	0.9	5.7	3.5	3.0	2.0
New Jersey	2.8	1.5	0.7	1.1	1.9	1.0	1.8	1.6	4.9	13.6
Massachusetts	1.6	1.1	0.9	6.1	4.9	5.4	2.1	1.8	1.6	2.0
Florida	3.1	2.3	1.3	1.1	0.9	1.0	0.4	0.4	8.5	7.9
Japan	11.6	15.4	17.7	16.7	16.1	33.8	32.2	7.3	8.1	11.5
China	7.9	20.5	17.8	10.3	5.9	2.5	5.3	6.3	4.6	3.1
Korea, Republic of	5.3	1.3	2.1	7.8	23.8	4.7	5.1	3.2	1.1	0.5
Germany	2.2	1.7	1.6	1.0	0.6	3.0	6.1	5.3	5.5	1.4
Taiwan	2.7	1.3	4.0	3.6	2.7	0.8	0.9	10.9	0.9	0.4
Netherlands	0.4	0.4	0.5	0.3	0.4	0.9	1.0	4.1	6.2	6.9
India	2.8	3.3	4.8	3.3	2.0	0.3	0.1	0.0	0.2	0.0
France	1.2	1.1	0.9	0.9	0.9	1.6	1.2	2.6	3.4	1.7
United Kingdom	1.1	1.0	1.0	0.9	0.7	2.5	2.2	1.6	1.3	1.3

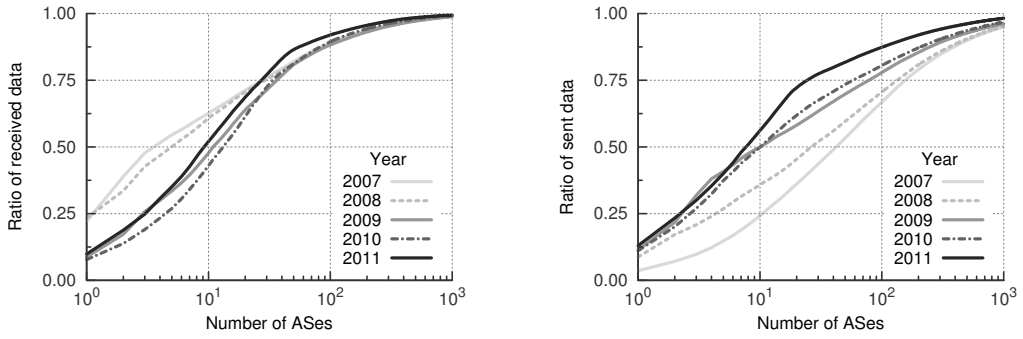
Table 5.3: Percentage of inbound and outbound traffic by country. U.S. state values are relative to total national traffic.

in figure 5.3b. European destinations overall have a small proportion of outgoing traffic, which appears to be shrinking over time. The most significant factor for the discrepancy between inbound and outbound traffic for Europe as a whole is the time zone difference, as traffic is measured at 05:00GMT. This however does not account for why outbound traffic overall has been falling. Since most outbound traffic towards Europe at the time of measurement is likely to be scheduled transfers with no human intervention, a plausible explanation for this trend is the gradual shift away from file-sharing using peer-to-peer applications. This is further corroborated by the rise of hosting solutions which facilitate file-sharing, as shall become evident when analyzing the breakdown of traffic by AS.

5.4.2 AS-level distribution

It has been widely noted that interdomain traffic has significantly changed over the past decade, with an increasing proportion of traffic flowing to and from a dwindling set of both large content providers and consumer networks. Such traffic consolidation is most apparent at the AS level, where a direct mapping to a commercial entity is forthcoming.

In the inbound direction traffic has remained consistently concentrated in the top 100 ASes accounting for approximately 90% of all data received, as shown by the cumulative distribution of inbound traffic by AS in 5.4a. There is a visible drop in concentration amongst the top ASes between 2008 and 2009 as a wide range of Japanese prefixes were rerouted through a different ingress. This shift is clarified in table 5.4, which lists the top ten ASes by received data for 2007, 2009 and 2011. While in 2007 traffic was



(a) (b)

Figure 5.4: CDF of (a) inbound and (b) outbound traffic by AS.

ASN	AS NAME	%
2914	NTT	29.92
36561	YOUTUBE	15.89
15169	GOOGLE	3.80
22822	LIMELIGHT	3.70
174	COGENT	3.03
9318	HANARO	2.46
3356	LEVEL 3	1.97
20940	AKAMAI	1.53
19166	ACRONOC	1.47
30212	DTI SERVICES	1.05

(a) 2007.

ASN	AS NAME	%
3462	HINET	9.78
15169	GOOGLE	8.64
43515	GOOGLE (YOUTUBE)	7.92
2914	NTT	5.89
46742	CARPATHIA (LAX)	4.17
4766	KOREA TELECOM	2.74
4134	CHINA TELECOM	2.62
3356	LEVEL 3	2.14
4837	CHINA UNICOM	2.10
36561	YOUTUBE	1.98

(b) 2009.

ASN	AS NAME	%
2914	NTT	10.79
20473	CHOOPA	8.86
43515	GOOGLE (YOUTUBE)	8.65
35415	WEBAZILLA	6.01
40824	WZ COMM.	4.79
15169	GOOGLE	4.66
40263	FC2	3.61
30212	DTI SERVICES	2.68
16265	LEASEWEB	2.56
29748	CARPATHIA (VA)	2.05

(c) 2011.

Table 5.4: Top 10 ASes for inbound traffic by year.

ASN	AS NAME	%
15169	GOOGLE	3.94
7132	SBIS AT&T	3.23
4134	CHINA TELECOM	3.14
10013	FREEBIT	2.91
4788	TM NET	2.37
9318	HANARO	2.21
9595	XEPHION NTT	2.09
9304	HUTCHISON AS	2.00
2914	NTT	1.90
4837	CHINA UNICOM	1.72

(a) 2007.

ASN	AS NAME	%
15169	GOOGLE	11.55
2510	INFOWEB	11.32
4134	CHINA TELECOM	9.10
2518	BIGLOBE NEC	6.00
3462	HINET	3.78
4837	CHINA UNICOM	3.01
14778	INKTOMI	2.04
7132	SBIS AT&T	1.51
36647	YAHOO	1.14
24560	AIRTEL	1.13

(b) 2009.

ASN	AS NAME	%
4766	KOREA TELECOM	11.39
15169	GOOGLE	8.66
2510	INFOWEB	5.93
3549	GLOBAL CROSSING	5.38
9318	HANARO	4.76
36647	YAHOO	4.63
2518	BIGLOBE NEC	4.23
46179	MEDIAFIRE	3.42
17858	KONYANG UNIV.	2.77
3462	HINET	2.68

(c) 2011.

Table 5.5: Top 10 ASes for outbound traffic by year.

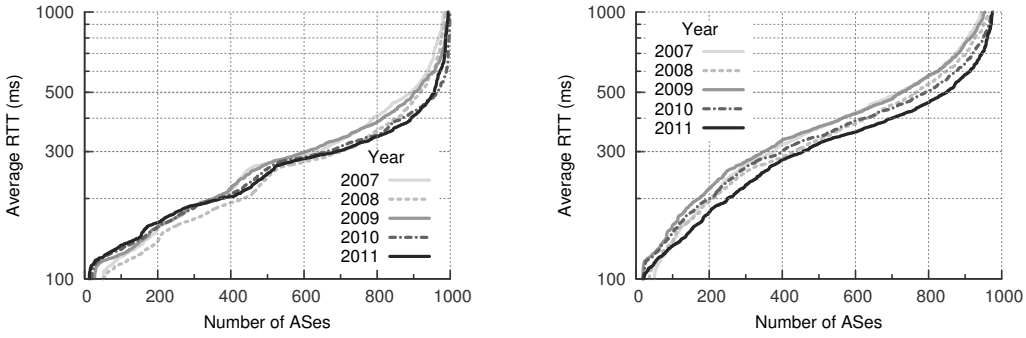


Figure 5.5: CDF of mean RTT by AS for (a) inbound and (b) outbound traffic.

already consolidated across a small set of ASes, a significant portion of transit traffic was still Asian: most traffic from NTT and Limelight originated from within Japan. Such traffic has gradually been pushed away from transit by 2011. Large carriers such as Cogent, Level3, Hanaro, China Telecom have also seen their importance diluted by ASes known to harbour OCH services such as Choopa, Webazilla, WZ Communications, Carpathia and LeaseWeb. Many of the hosted websites facilitate the distribution of copyrighted content, and as such are not capable of growing large enough to expand beyond hosted infrastructure without risking prosecution.

For outbound traffic, shown in figure 5.4b, consolidation has been much more perceptible. For the top 10 ASes alone, the proportion of traffic has more than doubled between 2007 and 2011. By 2011, the distribution of traffic amongst ASes for inbound and outbound traffic bears a striking similarity but the nature of this concentration is markedly different, as made apparent by table 5.5. Despite the increased importance of content providers and OCH services such as Google, Yahoo and Mediafire, significant portions of outgoing traffic continue to flow toward eyeball ISPs such as Korea Telecom, Infoweb, Global Crossing and Hanaro.

Overall, the observed traffic patterns match the insights provided by Labovitz et al. on the changing nature of interdomain traffic in [LIJM+10a], but highlight that such trends have occurred at different paces depending on the direction of traffic. Inbound traffic showed strong signs of concentration as early as 2007, whereas outbound traffic has only become dominated by large consumer networks and regional providers more recently. The implications for transit traffic from an Asian perspective is less intuitive: with the increased adoption of CDNs and IXPs, more transit traffic is being retrieved from further away as content in the United States shifts eastward.

5.4.3 Delay

While understanding where traffic flows to and from is of great value at an operational, commercial and often political level, it portrays a small part of a wider picture. For end-users it is of less concern where content is being retrieved from or routed through compared to how long it takes.

Intuitively delay should decrease over time as the Internet becomes more interconnected, resulting in less path stretch, and access technology improves, cutting down queuing delay in particular. This is

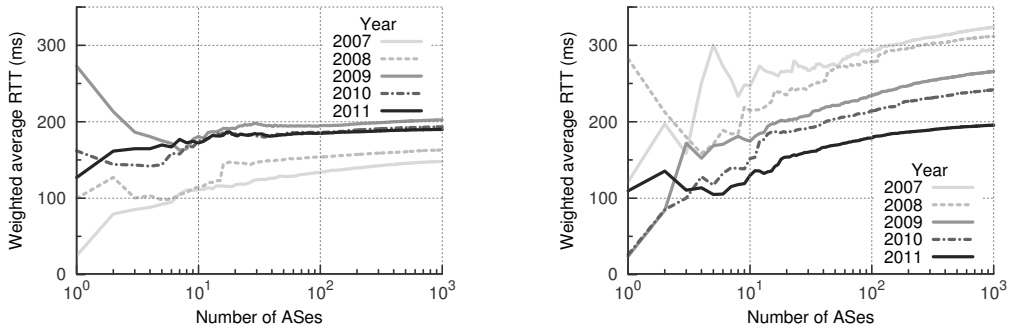


Figure 5.6: CDF of weighted RTT by AS for (a) inbound and (b) outbound traffic.

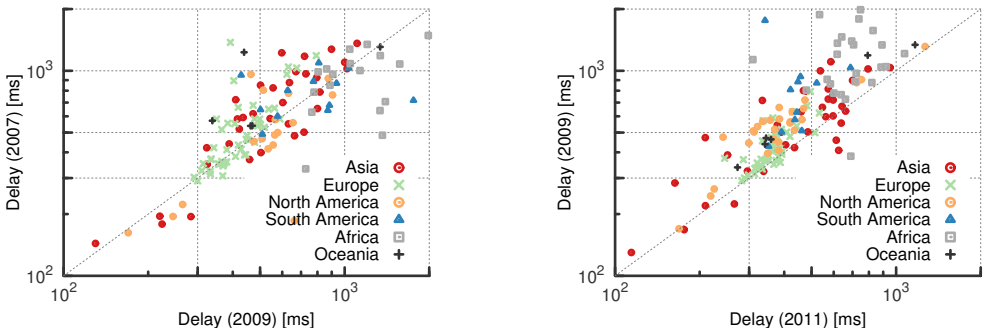
partially confirmed by figure 5.5, which displays the mean delay distribution for traffic in either direction. Given the long-tailed nature of traffic, many ASes have a limited number of RTT samples. As such, only the thousand most significant ASes in either direction are used to plot the respective CDF.

As with traffic distributions, the plots once again illustrate the same overall trend in subtly different patterns. While latency has dropped across the board, the rate of improvement is markedly different. Taking the median of both plots as a reference point, delay has dropped by $20ms$ between 2009 and 2011 for inbound traffic, nearly half the equivalent decrease for outbound traffic. The absolute values in both cases are still disparate: over 90% of ASes are reached within a round trip time of approximately $400ms$ when ranked by inbound traffic, whereas the equivalent value for outbound traffic is almost $200ms$ higher. For inbound traffic the average RTT is low enough that geographical properties are clearly visible. A first plateau close to $100ms$ is apparent for traffic to the American west coast, while traffic to European destinations is clustered close to $250ms$. Tellingly, this second plateau seems to be receding. When taken in conjunction with the geographic distribution of traffic presented in table 5.3 this seems to confirm the reduction in the number of sources within Europe.

A pertinent question at this point is in trying to understand how delay relates to traffic volumes. Given the different nature of stakeholders monopolizing traffic at either end of the spectrum, what can be said about the evolution of delay in either case? Figure 5.6 plots the cumulative distribution of the average RTT weighted by the respective volume of traffic. In interpreting such plots one should keep in mind that they provide a rough indicator of the average delay to be expected if one were to sample a packet belonging to the top N sources or destinations. As N increases, the resulting value approaches the average RTT for all traffic in a given direction.

Inbound traffic by AS highlights an expected rise in delay between 2008 and 2009, as both NTT and Limelight are replaced by more distant sources. However, from 2009 onwards the overall delay remains remarkably stable. While the cumulative distribution function of RTT shows improvement in delay at the tail, this results in very little improvement overall as traffic is dominated by a handful of entities.

Two explanations emerge for this behaviour. The first stems from the changing nature of the traffic being sampled. While functionally NTT represents the same entity over time, the traffic under obser-



(a) 2007/2009. (b) 2009/2011.

Figure 5.7: Scatter plot of mean RTT by country grouped by continent.

vation is very different. The local view of traffic has been stretched further afield as data has been increasingly exchanged over peering links, particularly at a national level. This is particularly noticeable in figure 5.6a, where the average delay towards NTT, the most significant AS for inbound traffic in both 2007 and 2011, increased by approximately $100ms$. This does not represent a degradation in quality of service, but rather a change in where traffic is flowing from within the AS.

A further reason relates to the placement of content. As previously established in table 5.3, there appears to be a migration of content away from California. OCH providers such as Lemuria, based in Florida, Mediafire, based in Texas and District of Columbia and Carpathia, based in Virginia, are all contained within the top 20 ASes and have shifted traffic further from locations which had traditionally benefited from low latency as viewed from Japan.

Analyzing the outbound traffic seemingly reveals the opposite effect, with the average weighted RTT dropping by over 100ms. Once more, there is no single reason which accounts for the entirety of this effect. In 2007 many of the top destination ASes were in developing Asian countries, where infrastructure has improved greatly since. Improvements in routing to countries such as China and the Republic of Korea have also had a positive net effect. This is visible in figure 5.7, where the average RTT aggregated by country is plotted. For clarity, countries for which RTT estimates are available for less than 50 days in a year are filtered out. Between 2007 and 2009 most Asian and European countries experience significant improvements in RTT. The minor exceptions are typically countries for which delay was already comparatively low. By 2011, latency to most countries had been reduced below $500ms$.

Finally, many of the very same companies which have had an effect of increasing RTT for inbound traffic, such as Mediafire or Ustream.tv, are also amongst the top destinations of traffic. It is interesting to note that as of 2011, data travelling from the top 1000 AS traffic sources was expected to experience the same latency as data travelling towards the 1000 most popular AS traffic destinations. In 2007, the value was two times higher for outgoing traffic. Traffic downstream is moving further away as content is not only placed closer to consumers and bypasses the transit link entirely, but also moves deeper within the United States, whereas traffic upstream has been drawn closer.

Chapter 6

TCP flow rate limitations

Providing a macroscopic view on where traffic originates from, and in what quantity, can be achieved by simply binning packets into flows and accumulating byte counts over geographical or topological locations. Uncovering application layer characteristics (i.e. how traffic is sent) is a more complex problem that requires additional methods to reverse engineer transport behaviour. This chapter builds on the analysis of the MAWI dataset presented in chapter 5, focusing on the longitudinal evolution of TCP behaviour. TCP plays a central role in mediating between application needs and network capacity. In the absence of congestion, TCP is responsible for increasing sending rates in a bid to make efficient use of available bandwidth. Conversely, should congestion arise, TCP is expected to reduce its rate. This form of congestion control embedded in TCP is largely credited with performing resource allocation across the Internet and averting congestion collapse. But to what extent does this behaviour describe TCP throughput in practice?

For some types of traffic, throughput is not strictly dictated by the outcome of congestion control. For one, inelastic traffic such as streaming media typically has bounds on the amount of capacity required. Beyond a certain throughput rate, bandwidth probing by TCP is often unnecessary and occasionally harmful. Since TCP drives itself relentlessly towards congestion, real-time applications may suffer from increased latency and jitter. Furthermore, content providers may wish to avoid exceeding the streaming rate for content which may not be consumed in its entirety due to user behaviour, for example channel hopping for video streaming services [RCG+10], or client limitations, such as buffering restrictions on mobile devices [RLL+11]. Such forms of *application pacing* are often also applied to elastic traffic. In some cases, high throughput is perceived and subsequently marketed as a value added service. One-click hosting services such Rapidshare and Megaupload [AMD09, SCBRSP12] actively monetise access to large amounts of content both through online advertising and subscription models to bandwidth tiers. Conversely, file sharing applications such as Bittorrent and other peer-to-peer clients allow users to rate limit transfers in order to reduce impact on competing traffic, or to provide incentives for participation [GCX+05].

Even beyond such application behaviour, flows may still be constrained by factors not pertaining directly to TCP congestion control. The transport layer is subject to strict bounds within which it can operate, potentially impeded by socket buffer sizes set by operating system (OS) vendors or tuned by

network administrators. There is an upper bound on the window size a receiver can advertise back to the sender; in the absence of windowscale negotiation [Bra89], no TCP connection can exceed a 64KB window. In addition, resource sharing is often subject to local policy. In the absence of adequate methods for readjusting how TCP distributes bandwidth, network operators and system administrators often trade efficiency for predictability, shaping traffic to conform to local notions of fairness or in anticipation for expected demand [KD11].

Given these different potential sources of rate control, *what can be said about their relative impact on Internet traffic at large*? Much of the underlying motivation is shared with the landmark study by Zhang et al. [ZBPS02] on the characteristics of Internet flow rates. Using traces spanning both access, peering and regional links, Zhang et al. analyse traffic according to potential rate limiting factors. Amongst other findings, host window limitations were found to affect over 30% of traffic for the access networks studied. Importantly, the authors found a strong correlation between flow throughput and flow size, postulating that this could derive from user behaviour, with large transfers more likely to be performed over higher bandwidth connections.

In addition to such general investigations, this chapter is equally indebted to comprehensive work of a narrower scope. Significant portions of the observed traffic pertain to well known applications which have been previously studied. Rao et al. [RLL+11] survey strategies used for video streaming at both Youtube and Netflix and characterise the properties of interleaved *block sending* patterns used to pace streams. These patterns are also the subject of [AN11], in which the burstiness of Youtube traffic in particular is found to result in considerable losses over residential connections. A large portion of the traffic observed in the MAWI dataset originates from HTTP file sharing services, commonly referred to as One-click Hosting (OCH) websites [AMD09]. In [SCBRSP12], the authors study the characteristics of such traffic over a three month period, detailing the different throttling strategies used by different providers.

6.1 Flow classification

The aim of this section is to describe a process which distinguishes those flows which have their throughput limited by mechanisms other than the usual TCP response to loss and delay. Each flow can be characterized as being either application paced, in which the sending application is limiting the data provided, host limited, whereby local constraints at either end host cap throughput, or receiver shaped, in which an artificial constraint is imposed by either a middlebox or receiver.

One fundamental precondition to decouple the influence that network loss, host configuration and TCP behaviour has on the throughput experienced by a flow is the reconstruction of the congestion window behaviour of TCP flows on the basis of observed data. Unfortunately, the congestion window value is internal to the sender's TCP state machine and may not manifest itself in the absence of sufficient data from the application layer. A more easily observed quantity which serves as a reasonable proxy for the congestion window is the number of unacknowledged bytes in flight, henceforth referred to as the *flight size*, which can be derived given an accurate estimate of the end-to-end delay. The evolution of both flight size and RTT can in turn be used to ascertain to what extent throughput is regulated by limitations

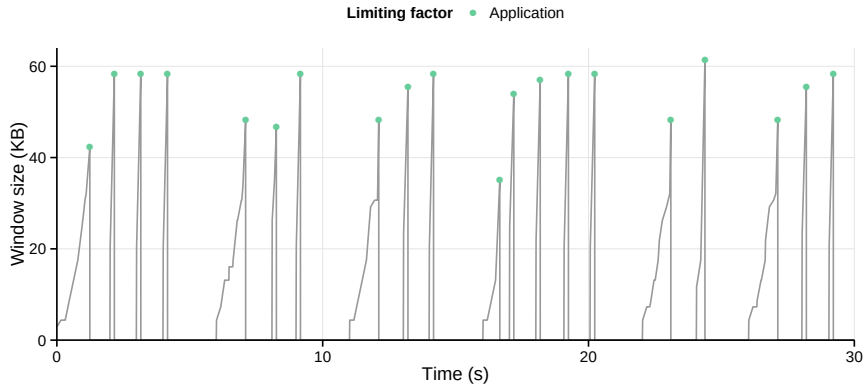


Figure 6.1: Congestion window over time for application paced flow.

imposed at different layers of the networking stack.

Given a stream of packets, the methodology presented in section 5.3 can derive a candidate RTT for a TCP flow. Given a candidate RTT, a stream of packets with arrival times $t_1, t_2, \ldots$ can be aggregated into a stream of *flights*. Intuitively, a flight is a clustered subset of a TCP flow which exhibits its own temporal coherence; alternatively, it can be thought of as a series of consecutive packets that were (roughly) generated by the sender as a response to the same protocol operation. A flight f_i that begins with the jth packet and ends with the kth is defined to have a *total flight time* $\tau_i = t_{k+1} - t_j$. The algorithmic selection of initial and final packets in such a way that the resulting flights are indicative of TCP behaviour remains an open problem. The RTT is assumed to provide a natural time frame for the operation of TCP. As such, given an initial packet π_j and an RTT estimate T, the kth (and final) packet is selected to minimise *the flight time error* $e_i = |T - \tau_i|$. This mechanism resembles the methodology described in [ZBPS02], but where flights are not defined as being both adjacent and disjoint; rather, flows are decomposed into a stream of potentially overlapping flights. This helps the algorithm mitigate the deleterious effects of small deviations in the estimated RTT, which alters the properties of each flight. Furthermore, since the flight size is continuous in time, it makes little sense to restrict window reconstruction to a single sample per round trip time.

Having obtained flight information from each flow, the predominant factor that affects its throughput can be determined. Within the context of TCP, flows are classified as being artificially constrained by three distinct processes: *application pacing*, *host limited* and *receiver shaping*.

6.1.1 Application paced

A flow whose throughput decreases because it has no outstanding data to send is temporarily limited by the application. Flights can be identified as being *application limited* if terminated with a packet smaller than the MSS and followed by an inter-arrival time greater than the RTT, as consistent with [ZBPS02]. The underlying reason for this defintion is that most TCP implementations will wait some time for subsequent bytes to be written to the socket if the next packet to be sent is smaller than the MSS, unless the TCP_NODELAY option is set [Nag84].

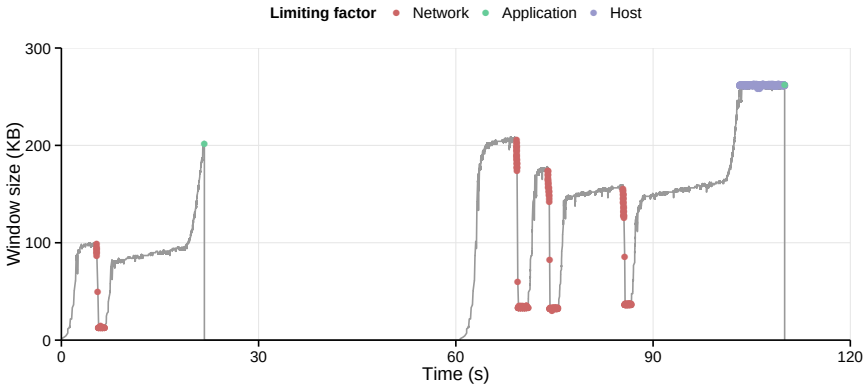


Figure 6.2: Congestion window over time for partially host limited flow.

A flow with *application limited* flights however is not necessarily *application paced*. In practice, all flows for which the final packet is observed contain at least one such flight. For the purposes of this work however, the focus remains on identifying cases in which throughput is predominantly determined by application behaviour. One such example is illustrated in figure 6.1, in which a stream is delivered by periodically writing blocks to the sending socket. The resulting network-level behaviour is distinct from traditional congestion control: short bursts are interspersed with protracted silence. Application limited flights, which terminate on non-MSS packets, are highlighted at the end of each burst.

This behaviour is in stark contrast to that exhibited in figure 6.2, where distinct transfers are multiplexed on top of a single transport association over time. From the perspective of the network, there is little to distinguish the behaviour of such traffic from independent TCP flows. Application paced connections such as Youtube traffic however exhibit a degree of regularity which can potentially be exploited by the network in predicting demand or smoothing bursts.

In order to identify such recurring behaviour, flows are classified as being *application paced* if the period between bursts terminated by *application limited flights* is consistently under 10 seconds and the standard deviation of the intermediate pauses is under one second. This definition in particular purposely ignores flows which exhibit long silence periods due to user interaction, and follows closely the behaviour historically associated to Youtube streaming in particular.

6.1.2 Host limited

Given sufficient bandwidth and traffic to send, a flow may encounter local constraints at either end-host which caps its throughput. For instance, the buffer space allocated on both the sender and receiver side is often pre-configured, and it is common practice to tune these values down on popular servers and managed infrastructure in a bid to conserve memory or bandwidth. A receiver is also limited in the window size it can announce to the remote sender; if the windowscale option [JBB92] is not negotiated during the TCP handshake, the advertised window cannot exceed 64KB.

In both cases, a local decision by either host can determine the upper bound of the flow rate. These *host limited* cases are characterised by a constant window size over time. The methodology described

for flight aggregation at the beginning of this section typically generates a large number of flights, representing many likely combinations given a base RTT estimate. In order to identify the flat-lined behaviour of a host limited flow, the flight stream is first filtered to remove some of the uncertainty derived from small fluctuations in the RTT. The maximum flight size observed for each RTT interval is then selected, with a sequence of flights being classified as host limited if the same maximum was observed over six consecutive RTTs (this is twice the period suggested in [ZBPS02]). In practice, increasing the period over which the maximum window size is tracked allows us to more accurately discern between host limited behaviour and more conservative bandwidth probing, such as that performed during the convex phase of TCP CUBIC [HRX08].

A flow may be host limited for only brief periods of its lifetime, as illustrated in figure 6.2. To filter out such cases where host limitations are not the predominant factor in defining flow throughput, a further requirement is imposed for a flow to be classified as being host limited: the average window size over a flow lifetime should be within 10% of the inferred host limit, which is not the case in figure 6.2. In practice, flows can exhibit both application pacing and host limitations, with bursts being sent at a capped window size followed by application pauses. In such cases, a flow will still be classified as being *application paced* if it meets the requirements set out in the previous section, as doing so provides evidence that it controls throughput in spite of the degraded performance provided further down the stack. This line of reasoning applies equally to the occurrence of sporadic loss; so long as block delivery is ensured within the time frame dictated by the application, it remains in control.

6.1.3 Receiver shaped

A flow which is neither *application paced* or *host limited* can still be artificially constrained by flow control (rather than by congestion control). Traditionally, in TCP the sender is responsible for regulating throughput. However, the receiver can also shape throughput by manipulating the *advertised window* announced on every acknowledgement. Such receiver window auto-tuning has been available on Windows operating systems since Vista [vis], and can also be leveraged by middleboxes to throttle inbound traffic [app]. To evaluate the potential impact of such behaviour, a further heuristic is proposed to identify receiver-shaped traffic. For flows in which both directions of traffic are observed it is possible to correlate the evolution of the advertised window with the size of reconstructed flights. Figure 6.3 displays an example of a receiver-shaped connection, in this case throttled by an intermediate middlebox. Since the advertised window may be fluctuating, it is not always obvious which of the many updates were effectively applied by the sender as successive values supersede each other. An example of a reconstructed flow which is subjected to receiver shaping is displayed in Figure 6.3.

For flows in which both directions are observed, it is possible to classify flights as being receiver-shaped if there is a statistically significant correlation between the advertised window size and the maximum flight size observed. Harnessing the stream filtering used in detecting host limited behaviour, such analysis is performed over a sliding window of 10 RTT intervals. A flight is flagged as being receiver shaped if the correlation between receiver window and flight size is statistically significant; a flow is considered to be predominantly receiver shaped if over half of its flights are flagged as such. This co-

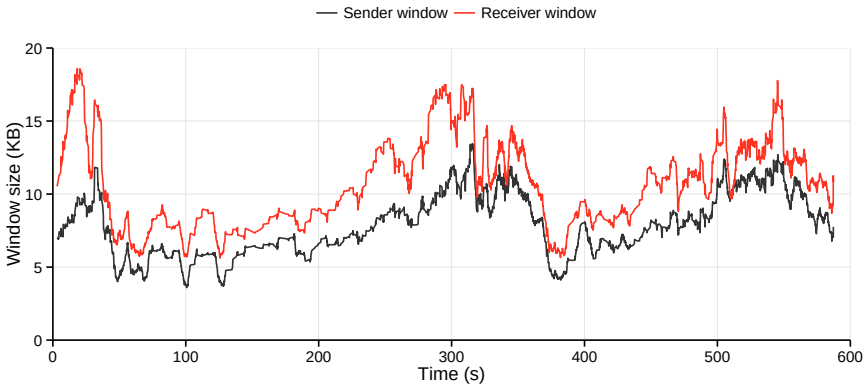


Figure 6.3: Congestion window over time for receiver limited flow.

variance analysis is not performed on flights which contain out-of-order or retransmitted packets. In such cases both the receiver and sender window sizes are correlated *by definition*: the receiver buffer will temporarily fill expecting the next packet in sequence while the sender will reduce its congestion window due to temporary setback in ACK clocking.

Given that receiver shaping classification requires correlating information in both directions of a TCP connection, it will come as no surprise that the absence of the reverse path can introduce false positives into our measurements. This happens because any given flow might be receiver shaped in such a manner that the heuristic erroneously attributes its behaviour to host limitations. In the absence of additional evidence, this misclassification is difficult to detect explicitly. Instead, the ratio of receiver shaped flows which would have been incorrectly identified is calculated for cases where the reverse path was not observed. This error rate can then be used to evaluate the accuracy of classifier results.

6.2 Revisiting assumptions

Having processed each daily trace individually, flow characteristics are aggregated longitudinally in order to trace the evolution of constraints affecting TCP across time and both spatial and topological dimensions, following the process described in section 5.2. The ensuing analysis revisits four commonly held assumptions regarding Internet throughput. The aim in doing so is to provide a much needed factual verification of these assumptions, which itself can lead to a re-appraisal of Internet throughput modelling efforts, particularly those enunciated in Zhang et al. [ZBPS02]. Although all models require simplifying assumptions in the name of analytic tractability, the primary goal of this section is to inform on which are the best assumptions to make if one is setting out to use or develop an Internet traffic throughput model.

6.2.1 Throughput is primarily shaped by TCP

Internet flow rates are commonly viewed as the output of congestion control embedded at the transport layer. While it is often convenient to model flow throughput according to the steady state behaviour of such algorithms, there are many potential caveats. For one, there is an implicit assumption that the

Year	Loss (%)	Limitation (%)			
		Application	Host	Receiver	Total
2007	1.29	49.47	18.58	0.55	**68.60**
2008	1.37	49.55	17.80	0.69	**68.04**
2009	1.44	47.10	14.50	2.57	**64.17**
2010	1.22	36.78	20.44	3.21	**60.43**
2011	0.82	46.10	13.49	0.60	**60.20**

Table 6.1: Percentage of traffic bytes affected by each constraint by year.

network is the bottleneck. Under such conditions, TCP acts as a distributed optimisation algorithm in allocating capacity to flows. Section 6.1 however presented several cases where such an assumption does not hold. The prevalence of application pacing, host limitations or receiver shaping can all condition the accuracy of models which assume only elastic traffic adjusting to network conditions alone.

Table 6.1 displays the extent to which each of these limitations affects inbound traffic in the MAWI dataset over time. The bulk of the volume in bytes is either conditioned by host limits or application pacing. The use of receiver shaping on the other hand is both small in scale and temporally confined to 2009 and 2010. Over five years, the overall effect of the three selected constraints has dropped by close to 10%.

To understand where these dynamics stem from, table 6.2 further breaks down these findings by autonomous system, listing the effect of each limitation for the five most significant traffic sources per year. Over the observed five years, traffic remains similarly consolidated: approximately 90% of all inbound traffic is sourced from the top 100 ASes. However, the weight of the most significant sources changes considerably. In 2007 and 2008, a considerable proportion of the traffic exchanged over the interdomain link was content hosted within Japan (NTT, Limelight). From 2009 onwards, most of these local sources established peering connections, bypassing the observed link entirely. This accounts not only for the significant drop of traffic from NTT, but also its altered nature: after 2009 traffic from NTT travelled from further away and was less likely to be application paced.

As the weight of traditional carriers such as Cogent and NTT has waned, ASes known to harbour one-click hosting services such as Choopa, Webazilla, WZ Communications, Carpathia and LeaseWeb have gained significance. Since many websites hosted in these ASes facilitate the distribution of copyrighted content, they have an incentive to continue using hosted infrastructure rather than deploying their own and risking prosecution. Furthermore, these domains are more likely to host applications which resort to capping the maximum window size as a means of throttling traffic. The increased weight of ASes which resort to these methods, such as Red Hat and Carpathia, significantly contributes to the unexpected increase of host limitations for 2010 displayed in table 6.1.

Overall, these findings demonstrate that flow rates are not typically dictated by TCP congestion control alone. While the impact of application pacing and host limitations on rate control is decreasing, as of early 2012 60% of all inbound traffic was still subjected to either. The reduction of application pacing however may reflect the nature of the observed link, as many traditional streaming

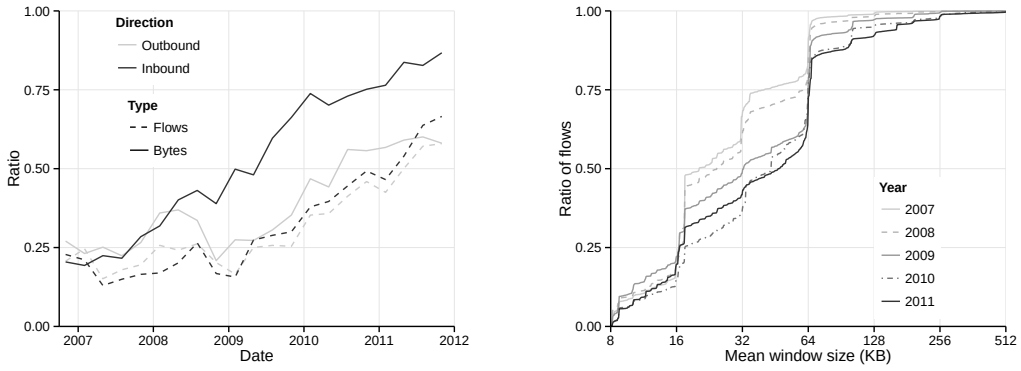
YEAR	ASN	AS NAME	TRAFFIC (%)	LIMITATION (%)		
				APPLICATION	HOST	RECEIVER
2007	2914	NTT	28.34	65.29	14.68	0.39
	36561	YOUTUBE	15.16	77.41	11.19	0.11
	22822	LIMELIGHT	8.12	55.11	21.90	1.37
	15169	GOOGLE	3.72	24.11	10.29	0.08
	174	COGENT	2.87	47.65	32.22	0.76
2008	2914	NTT	17.16	62.40	13.71	1.03
	22822	LIMELIGHT	10.57	65.40	21.48	0.52
	36561	YOUTUBE	9.15	72.48	12.16	0.09
	2518	BIGLOBE NEC	5.06	84.34	5.84	0.10
	15169	GOOGLE	3.52	52.98	17.13	0.18
2009	3462	HINET	9.93	60.07	4.82	0.05
	15169	GOOGLE	8.78	74.79	12.16	0.02
	43515	GOOGLE (YOUTUBE)	8.08	83.46	9.83	0.14
	2914	NTT	5.69	39.76	8.37	0.16
	46742	CARPATHIA (LAX)	4.27	41.04	48.01	2.03
2010	2914	NTT	7.39	21.80	4.91	0.00
	31976	RED HAT	7.03	9.62	41.63	0.00
	7366	LEMURIA	5.88	51.95	15.72	5.85
	43515	GOOGLE (YOUTUBE)	5.22	77.76	8.41	0.14
	46742	CARPATHIA (LAX)	4.69	33.06	42.71	4.21
2011	2914	NTT	10.37	50.33	8.19	0.18
	20473	CHOOPA	8.92	54.03	19.24	0.21
	43515	GOOGLE (YOUTUBE)	8.69	69.71	7.56	0.16
	35415	WEBAZILLA	6.05	40.02	11.23	0.95
	40824	WZ COMM.	4.83	42.08	17.43	0.05

Table 6.2: AS-level analysis of throughput limiting.

providers have migrated towards peering or CDNs, bypassing interdomain links entirely. As such it is reasonable to expect the effect of application pacing to be more pronounced when considering traffic beyond transit. By 2011, successive capacity upgrades have led to a less congested network, but one where predicting how bandwidth is shared is fundamentally harder due to the influence of stakeholders such as content providers and operating system vendors.

6.2.2 Throughput is primarily sender driven

A more widely held and less frequently enunciated assumption is that flow throughput is primarily determined at the sender side. Intuitively, it is in a receiver's best interests to maximize the flow rate, while the sender bears the responsibility for sharing network capacity and reducing the overhead incurred due to losses. The Internet architecture however confers the receiver the ability to throttle rates through flow control. From answering the previous assumption, it is clear that throughput *is* mostly determined by the actions of the sender: receiver shaping and host limitations together affect at most 24% of all traffic.



(a) TCP windowscale deployment over time.

(b) CDF of window sizes for host limited flows.

Figure 6.4: Longitudinal evolution of TCP window parameters.

Despite this it is worth understanding the nature of these host limitations in particular, and towards which direction flow control is swinging.

A critical component in determining the upper bound for the congestion window size is the negotiation of the TCP *windowscale* option during the initial handshake. In its absence, a sender cannot have more than 64KB in flight. Furthermore, the *default buffer size* on either end of the connection can also limit the size to which the congestion window can increase. Both settings are primarily subject to operating system configuration. Throughput conditions on the receiver side improve as OS upgrades are rolled out. The installed user base within WIDE is comparatively stable over time, and as such is expected to exhibit continual improvements unless a significant OS rollback were to occur or a large set of users with outdated OSes were to be added to the network. This however is unlikely, and a more plausible explanation for any significant degradation in host limitations lies in macroscopic shifts in routing or application popularity which lead to a change in *where* traffic originates from.

This hypothesis is tested by first verifying windowscale deployment over time. Figure 6.4a shows the ratio of traffic and flows for which windowscale was successfully negotiated. Results are calculated solely over traffic where the initial handshake was observed. For added context, data for the outbound direction is also displayed. The first result that stands out is the steady increase over time of windowscale usage, rising from 25% of all inbound bytes in early 2007 to almost 80% by late 2011. Furthermore, the effects of content consolidation manifest themselves in the disproportionate coverage of bytes when compared to flows. With the reduced stake of large ISPs in inbound traffic, transit traffic has become dominated by a small set of centrally managed stakeholders such as Google, lowering the effective barrier for deployment of protocol extensions. Conversely, the temporary drop in windowscale adoption for inbound flows in 2009 is due to the increase of traffic from Asian sources, in particular HiNet.

Given the prevalence of windowscaling, the primary source of host limitation should therefore be the configuration of socket buffer sizes. Figure 6.4b shows the distribution of the average window size for flows which are flagged as being host limited. While the 64KB limit intrinsic to TCP is a common upper bound on window size, other defaults are apparent and have shifted over time. The

	Total (%)		Receiver (%)	
Year	Flows	Bytes	Flows	Bytes
2007	3.86	20.31	70.82	74.33
2008	3.91	19.26	68.96	71.16
2009	2.60	15.29	61.54	62.81
2010	3.07	21.73	53.16	64.63
2011	1.76	14.11	51.27	60.82

(a) Inbound traffic.

	Total (%)		Receiver (%)	
Year	Flows	Bytes	Flows	Bytes
2007	8.22	24.24	76.50	82.54
2008	11.80	32.40	68.81	84.38
2009	10.40	30.50	62.50	84.28
2010	4.00	27.00	76.14	88.07
2011	3.01	25.94	72.00	85.91

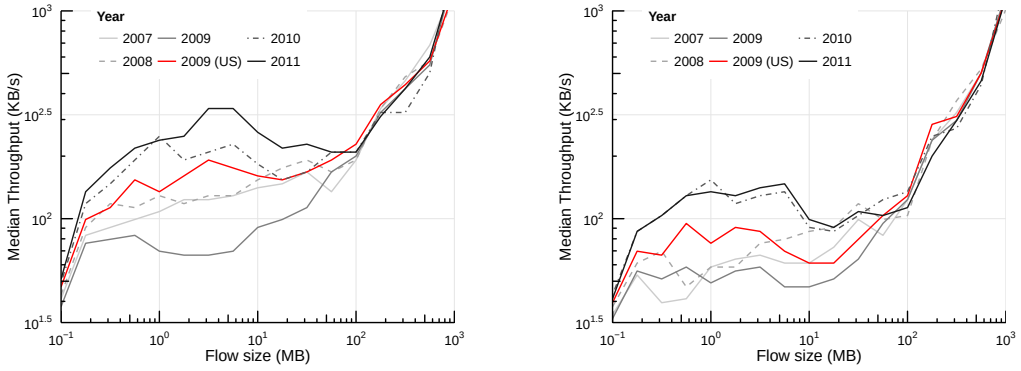
(b) Outbound traffic.

Table 6.3: Percentage of host limited traffic over time by total number of flows and bytes. The proportion for which the receiver side was the bottleneck is also shown.

use of 16KB and 32KB buffer sizes (default buffer sizes for Windows XP and Vista respectively) was progressively phased out over the five year period. In addition to traditional power-of-two increments of the window size, different limits are apparent amongst hosting providers: 50KB, 100KB (The Planet), 160KB (Limelight) and 200KB (SoftLayer), reflecting the overall weight such ASes can have in shaping transit traffic. The influx of Asian traffic in 2009 led to an increase in observed host windows beneath 16KB.

While figure 6.4b demonstrates that host limitations for inbound traffic have been lifted over time, it still does not adequately answer on what side of the connection they are imposed. Table 6.3 breaks down the proportion of host limited traffic over time for both inbound and outbound direction. In addition to presenting the ratio of flows and bytes affected by host limitations, the relative proportion of traffic identified as being conditioned by the receiver side is also displayed. In either direction a very small fraction of flows are affected. Small flows are both numerous and unlikely to last long enough for window limits to be reached or reliably detected. The affected flows therefore tend to be large, and as such can translate into a significant amount of traffic. The proportion of flows and bytes for which the receiver side imposed the maximum window size dropped by 20% and 15% respectively over five years for inbound traffic, reflecting the successive OS upgrades performed for hosts within WIDE. Interestingly, these trends do not surface for outbound traffic: hosts outside Japan were consistently more likely to dictate the maximum window size. In part, this reflects the different nature of the traffic under observation: outbound traffic for this dataset is more geographically and topologically diverse, with content in many cases being retrieved from Japan by residential hosts from within Asia.

The endpoint which ends up dictating the maximum achievable throughput through flow control is typically a function of the OS adoption cycle. With the windowscale option covering 80% of all inbound traffic, the main source of host level constraints are now conservative buffer sizes. For this dataset, hosts internal to WIDE have seemingly been upgraded at a faster rate, or less conservatively, than their remote counterpoints. As such, throughput has become increasingly sender driven over time for inbound traffic.



(a) Mean congestion window over mean RTT.

(b) Flow size over flow duration.

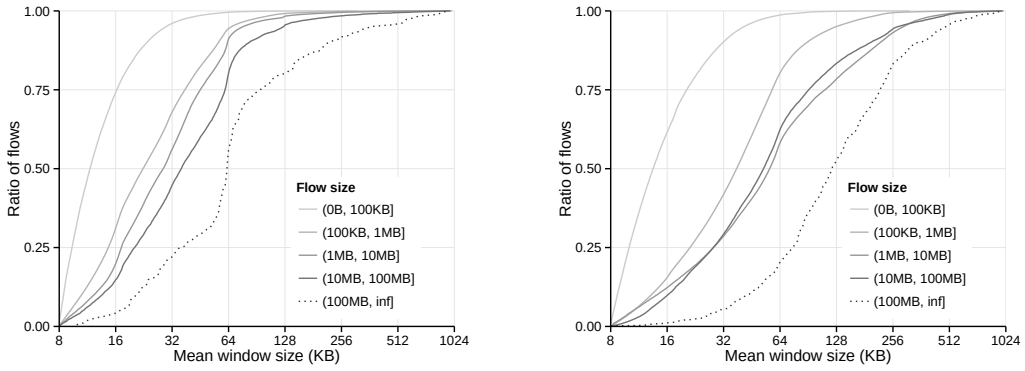
Figure 6.5: Median throughput for inbound traffic by flow size.

6.2.3 Throughput is correlated with flow size

Given host limitations are most likely to affect large flows, it's worth considering whether other constraints are applied disproportionately across flow sizes. A commonly held assumption is that throughput is correlated with flow size, which has been verified empirically in previous studies [CAL+13, ZBPS02]. Much of the data used in these studies however precedes the widespread adoption of high bandwidth connections and use of streaming media, both of which can impact the extent to which contention occurs in the network.

Figure 6.5 shows the median throughput as a function of flow size, by year. In figure 6.5a, flow throughput is calculated as the ratio between the mean TCP window size (in bytes) and the mean flight length (in seconds). Compared to the more commonly used ratio of flow size by flow duration, displayed in figure 6.5b, this method is less susceptible to application behaviour and as such provides a more accurate estimate of the achievable rate. In both cases, flows are binned by size on a logarithmic scale, with median throughput calculated across each bin. Due to routing changes and increased congestion, overall throughput in 2009 is lower than other years given there is a greater proportion of traffic from Asian neighbours, particularly over smaller flow sizes. For reference the throughput for traffic from the US alone is plotted for 2009,in which case a more natural yearly progression becomes apparent. For both plots, a clear disparity is visible across flows sizes: for flows in the 10MB to 100MB range, although throughput has consistently increased with time, it has done so at a lower pace than for flows under 10MB.

These results confirm the notion that the highest throughputs are attained by the largest flows, but they also show that improvements in throughput do not apply equally to all flow sizes. Whereas throughput has consistently improved for low-volume traffic, it has not done so for high-volume traffic. Hence, these findings suggest an increased differentiation between high-value, low-volume traffic whose throughput has markedly increased, and low-value, high-volume traffic whose throughput has stagnated. Although the reported flow sizes are not a reliable predictor of the overall traffic volume amassed over a flow's lifetime given the short time span of each daily trace, this seemingly subverts the notion that



(a) 2007.

(b) 2011.

Figure 6.6: CDF of the average window size by flow size by year.

flow rates are strongly correlated with flow size in a simple, proportional fashion. This is expounded by further analysing the average window sizes across flow sizes, displayed in Figure 6.6. In 2007, there is a visible correlation, with larger flows attaining higher window sizes. Furthermore, the distributions cluster prominently around 64KB due to a low rate of windowscale negotiation. By 2011, this clustering is less pronounced, with window sizes increasing across the board, but with larger flows often outpaced by shorter counterparts.

Clearly, the extent to which rates are constrained is closely tied to flow size. Table 6.4 breaks down the results from table 6.1 by flow size. Most limitations will invariably affect larger flows, as applications which shift more traffic, such as streaming media or bulk file transfers, are more likely to either attain or self-impose constraints on flow throughput. Many small flows on the other hand never exit slow start, in which case none of the studied constraints will be reached or readily identified. This dichotomy is reflected on loss rates, which will be higher for flows regulated by TCP congestion control. Additionally, the discrepancy in loss rates is further exacerbated by geographic properties: traffic exchanged over poor infrastructure tends to be smaller, with flows from China exhibiting particularly high end-to-end loss rates.

These results suggest that network upgrades are unlikely to improve performance for significant proportions of traffic. This is most visible in figure 6.5, where improvements in capacity for coping with higher bursts of activity (figure 6.5a) has outpaced the actual delivery rate set by applications (figure 6.5b). Given the popularity of emulating a constant bit rate service over TCP, that no such abstraction is provided at the socket level API is unfortunate.

6.2.4 Throttling primarily affects heavy hitters

The observations so far have highlighted that flow throughput is often subjugated from TCP by external stakeholders. A third important element in the ensuing tussle is in understanding the role operators can play in imposing their own preferences upon traffic. As such, this section provides a brief overview of the extent, and under what circumstances, customer networks resorted to receiver shaping over the duration of the dataset. From preliminary inspection of table 6.4, it is apparent that receiver shaping was

YEAR	LOSS (%)	LIMITATION (%)			
		APPLICATION	HOST	RECEIVER	TOTAL
2007	1.90	14.65	5.45	0.10	**20.20**
2008	2.27	14.99	5.37	0.09	**20.44**
2009	2.39	15.66	3.83	0.55	**20.03**
2010	2.15	10.55	4.18	0.36	**15.09**
2011	1.19	11.13	2.53	0.05	**13.71**

(a) Flows under 10MB.

YEAR	LOSS (%)	LIMITATION (%)			
		APPLICATION	HOST	RECEIVER	TOTAL
2007	0.96	61.62	23.07	0.71	**85.40**
2008	0.88	61.49	21.94	0.92	**84.35**
2009	0.98	57.86	17.70	3.28	**78.85**
2010	0.71	43.97	24.45	4.03	**72.45**
2011	0.62	52.95	15.55	0.71	**69.21**

(b) Flows over 10MB.

Table 6.4: Percentage of traffic in bytes affected by each constraint by year according to flow size, along with aggregate retransmission ratio.

limited in both scope, affecting at most 4% of bytes, and time, being primarily concentrated within 2009 and 2010. A breakdown of receiver shaping by traffic source is provided in table 6.5, listing for each year the five most affected stakeholders within the top twenty ASes, and their respective contribution to the overall traffic and retransmissions observed yearly.

Prior to 2009, receiver shaping mostly targeted flows which attained the highest throughput, and may have in part been performed by hosts. In addition to affecting Microsoft, which distributes Windows updates using the same flow control mechanisms exploited by middleboxes, some CDN traffic was also reined in. By 2009, and likely as a reaction to increased contention within their networks, the network-level footprint of receiver side throttling shifts from shaping by rate, to shaping by volume. Specific targets start to emerge within OVH, Cogent and Level 3 and TeliaNet, all of which hosted significant file-sharing websites at the time which would continue to be affected throughout 2010. By 2011, receiver shaping mostly subsided, affecting primarily Lemuria (HotFile) and Mediafire. In the presence of increased contention, some customer networks felt obliged to curb traffic which in many cases was already limited by the source. The selected targets of shaping however were neither the biggest contributors in terms of volume, nor the most aggressive senders as reflected by the relative proportion of retransmissions, most likely being singled out instead based on the perceived legality of the content downloaded. When consulting the overall popularity of these targets in table 6.2, it is apparent that some of the most throttled ASes such as Carpathia and Lemuria had been far more popular in previous years, suggesting that users may migrate to other content providers when confronted with lower rates. While successive bandwidth upgrades alleviated the need for throttling, it is unclear whether the middleboxes

responsible were discontinued, or limits were merely raised to the extent where the sender side would once again become the bottleneck.

In some cases, multiple stakeholders apply different rate control mechanisms simultaneously, leading to quality degradation and unusual rate behaviours. Conventionally, it is assumed that it is in the best interest of content providers to use network resources as fully as possible, whereas ISPs have it in their best interest to police resource usage and control flow rates. However, as shown here, current business practices can create an alignment of incentives where both content providers and ISPs choose to limit the throughput of a particular class of traffic. This may lead to traffic suffering from artificially slow rates that are much worse than those experienced by other classes. In these cases, the combination of multiple rate control techniques being applied by different stakeholders may lead to a traffic profile whose rate behaves in a manner quite removed from the commonly assumed TCP dynamics.

6.3 Conclusions

The focus of this chapter has been on elucidating the main factors that affect flow throughput, but which escape traditional TCP modelling based on end-to-end loss and delay. In particular, the changing role of *host limiting*, *application pacing* and *receiver shaping* in defining flow rates across five years of transit traces is explored. The results show that for the observed link, over *half* of all inbound TCP traffic can be ascribed to one of the aforementioned constraints. Furthermore, continuing OS upgrades are responsible for having progressively lifted the artificial throughput constraints imposed by the host stack. In particular, windowscale negotiation for inbound traffic increased threefold in the observed period, covering over 80% of all observed bytes by 2012; in addition, buffer sizes have also shown continuing increases over time.

These developments have significantly improved throughput, in particular for smaller flows. However, evidence of throughput limiting effects independent from available end-to-end capacity was also uncovered. This means that no amount of bandwidth will directly improve TCP rates for a considerable amount of traffic. Application-driven techniques for chunked transfer in particular are widely used, accounting for 40% of all inbound traffic observed in 2011. Finally, there is significant evidence of receiver traffic shaping prior to 2011 based on the modification of the receiver advertised window in a bid to curtail congestion.

Year	ASN	AS name	Receiver shaped (%)	Contribution (%)	
				Bytes	Rexmt
2007	8071	Microsoft	4.61	0.69	0.17
	41690	DailyMotion	2.57	0.52	0.13
	20940	Akamai	1.50	1.49	0.90
	22822	Limelight	1.37	8.12	5.01
	19166	ACRONOC	0.81	1.40	1.22
2008	21844	The Planet	2.43	0.76	0.82
	174	Cogent	2.02	1.89	1.39
	19166	ACRONOC	1.61	2.02	1.16
	1299	TeliaNet	1.25	1.25	1.59
	2914	NTT	1.03	17.16	13.24
2009	16276	OVH	31.97	1.26	0.29
	3356	Level 3	6.24	2.18	1.50
	22822	Limelight	5.92	1.60	0.88
	174	Cogent	5.49	1.30	0.91
	16265	LeaseWeb	5.09	1.98	1.06
2010	1299	TeliaNet	25.67	1.31	1.43
	3356	Level 3	17.33	2.71	3.16
	16276	OVH	15.28	1.84	0.32
	16265	LeaseWeb	7.70	3.23	2.16
	29748	Carpathia (Ashburn)	7.46	2.62	1.30
2011	7366	Lemuria	4.45	1.96	1.56
	46742	Carpathia (LAX)	1.89	1.64	0.68
	46179	Mediafire	1.20	1.54	0.82
	29748	Carpathia (Ashburn)	1.08	2.06	1.27
	35415	Webazilla	0.95	6.05	3.33

Table 6.5: AS-level analysis of receiver shaping. For each year, the five most affected ASes are listed with the total proportion of traffic which was receiver shaped as well as the overall contribution of the AS to the total volume of bytes and retransmissions for that year.

Chapter 7

Network support for transport resilience

This chapter refines the concepts proposed in PREFLEX given the insights provided by the MAWI dataset presented in chapters 5 and 6. This evidence-based approach is further enhanced by adapting the resulting architecture, INFLEX, to function within existing frameworks for Software-defined networking (SDN). This chapter is organized as follows:

Section 7.1 reviews results from the MAWI dataset and their implications for the design of a resource pooling architecture.

Section 7.2 provides an overview of SDN and OpenFlow, a standardized protocol for interfacing between control and data plane.

Section 7.3 describes the design of INFLEX, a cross-layer architecture for network resilience.

Section 7.4 evaluates path fail-over and network overhead of the proposed solution

Section 7.5 discusses how INFLEX can be extended from providing resilience alone into a unifying traffic management solution, encompassing most of the benefits put forward in chapters 3 and 4.

7.1 Design considerations

In chapter 3, one of the primary justifications for network assisted resource pooling was that existing solutions, such as MPTCP, were not suitable for most flows. This assumption was founded on the fact that most flows were too short to make effective use of available bandwidth without prior knowledge of path quality. The analysis in chapter 6 paints an even starker picture than was originally assumed: significant portions of traffic are limited and as such not inclined to make efficient use of capacity even if it were available. With over half of all TCP traffic constrained by factors other than loss, the allure of MPTCP as a means of making use of all available end-to-end capacity between endpoints seems circumscribed to a small set of high-throughput transfers.

This suggests that the flexibility and robustness provided by resource pooling may be a more compelling feature than efficiency alone. Despite being broadly designed for robustness, the current Internet architecture remains remarkably vulnerable to failures. Managing faults still poses a significant operational challenge, in part because faults can occur at every layer of the networking stack, and can affect any

element along a network path. PREFLEX was designed with efficiency in mind, addressing resilience in passing. In particular, PREFLEX expects sufficient host feedback in order to adjust path weights. This proves adequate for when many flows experience the same outage, but may fail to detect faults which affect isolated flows. This chapter instead formulates a resource pooling architecture by first dealing with the single case – addressing when a transport protocol detects failure – and then extrapolating to the domain level, providing efficient traffic management.

7.1.1 Latency

While efficiency has become a less pressing concern for most traffic, latency has become increasingly valued, particularly as providers attempt to ensure responsive, interactive services. This trend manifests itself in two manners within the MAWI dataset. Firstly, chapter 5 detailed how significant volumes of traffic have been progressively shifted from transit to local peering links. Given propagation delay is a critical component of end-to-end delay, putting content closer to end-users is an extremely effective method of cutting latency, and has assisted content distribution networks in becoming a key stakeholder within the Internet. Secondly, flow rates for shorter transfers are increasing at a faster rate than for large flows, as displayed in figure 6.5. Developments such as the increase of the initial window size [DRCC10], TCP Fast Open [RCC+11] coupled with higher socket limits and windowscale negotiation highlighted in chapter 6 have all benefited shorter flows disproportionately.

The original design of PREFLEX imposes a potentially significant burden on latency by requiring network intervention at each flowlet. While such network processing is small, and will necessarily improve over time, in hindsight tying together path selection and flowlet start does not feel entirely justified. Chapter 6 demonstrated that over 40% of all traffic is regularly application paced, and so the prevalence of opportunities for balancing at the flowlet level is unquestionable. However, imposing network intervention on each flowlet start is excessive for short, delay-sensitive traffic, particularly since the benefit such small-grained balancing provides the network is negligible. Requiring every PREFLEX router to inspect packets on the basis of packet markings made by external entities was also a potential source of Denial-of-service (DoS). While such attacks are easily dismantled by rate limiting the number of FNE packets received across network boundaries, at the very least this represents additional overhead for the operator, who must adequately configure and manage such a service. As such, in considering an architecture for resilience, it would be beneficial to both instantiate network processing only when required and be robust to DoS attacks by design.

7.1.2 Deployment

Ideally resilience could be implemented at the transport layer alone, for the same motives rate control is best left to end-hosts: ultimately, the host is best positioned to detect end-to-end path faults and can often react over shorter time scales than the network, which must concern itself with reconvergence. This approach for path fail-over was a significant feature in Stream control transport protocol (SCTP) [Ste07]. Unfortunately, deployment of SCTP has been negligible in over a decade since standardization, in part because the pervasiveness of middleboxes has significantly affected the ability for new transport protocols to be deployed. More recently MPTCP has been proposed addressing many of the same concerns

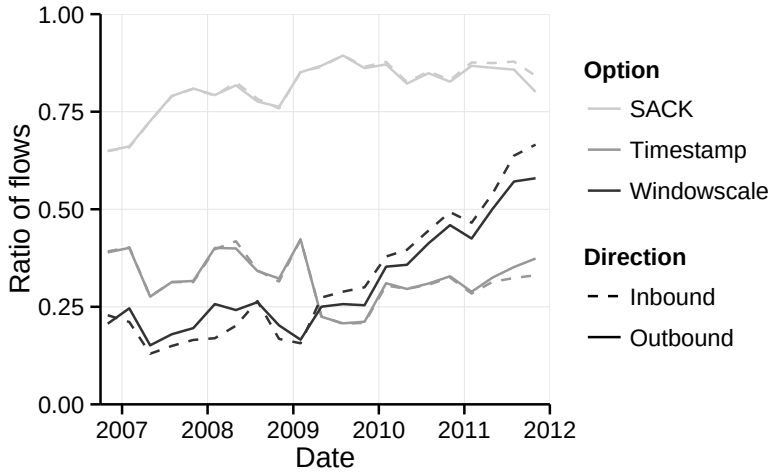


Figure 7.1: TCP option usage.

as SCTP whilst maintaining the same wire format as TCP, thereby ensuring middlebox compatibility. Despite this, widespread deployment is far from guaranteed, and is largely tied to the rate of OS adoption as evidenced in chapter 6.

As a reference point, figure 7.1 tracks the use of three TCP options by overall volume in flows across both directions in the MAWI dataset. While Selective acknowledgement (SACK) is successfully negotiated for most connections, the deployment of the timestamp and windowscale options has lagged. The former primarily assists in providing more accurate RTT estimates and is therefore largely auxiliary. The latter however is critical for performance: without windowscale negotiation, a sender's congestion window cannot exceed 65KB. Despite offering a clear benefit to both endpoints, being simple to implement and incurring a low overhead, windowscale deployment has only recently picked up momentum, two decades since standardization. Expecting substantial deployment of a more complex and costly extension such as MPTCP over the near future is likely optimistic. Critically, transport extensions require receiver adoption and are therefore subject to the willingness and ability of users to upgrade their OS.

This shortcoming affects PREFLEX in equal measure. The original assumptions for deployment in chapter 3 were consistent with an Internet where traffic was mostly exchanged between peers. Currently however, the asymmetry between the software which runs on either endpoint has never been greater, in large part due to the proliferation of mobile devices. The barrier to deployment has been raised further: receiver side deployment of even modest TCP extensions can be protracted, even when incentives are aligned. Any proposed modification must find its way upstream to the code base of different OSes and be verified to perform adequately over a greater variety of constraints than in the desktop computing era. Rather than proposing a path for incremental deployment, this chapter focuses instead on how to obtain similar benefits immediately – modifying sender side hosts only.

7.1.3 Multipath routing

A host, however, cannot directly affect routing without changing destination address, which would break legacy TCP receiver side implementations. Additional extensions are required on the sender side network

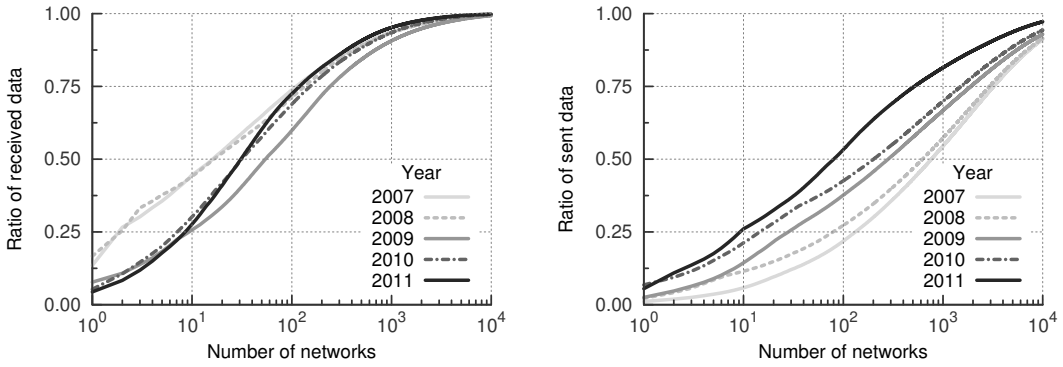


Figure 7.2: CDF of traffic by announced network prefix for (a) inbound and (b) outbound traffic.

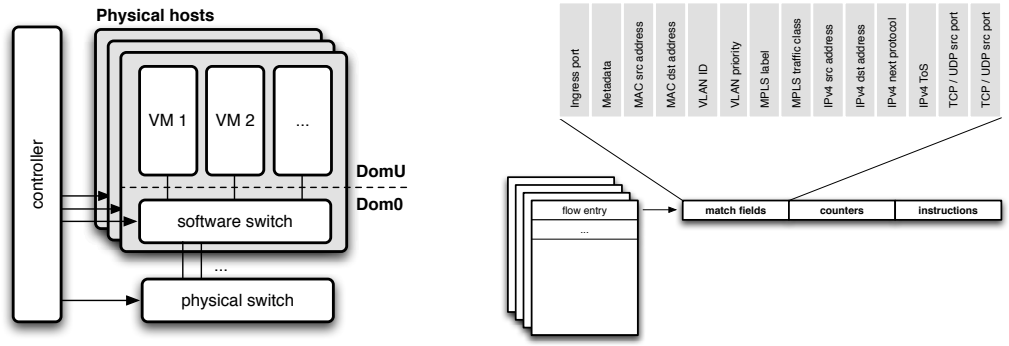
to enable multipath forwarding. Conventional wisdom suggests that maintaining parallel routing planes requires a proportional increase in table size [WHPH08], which itself can be subject to exponential growth. In practice however, this state can be significantly reduced by forsaking coverage for a small proportion of traffic. Rather than reflect the entirety of its potential path diversity for all traffic, an edge provider can instead provide additional routing planes for only a subset of popular prefixes.

The extent to which such a gain is possible for the MAWI dataset is quantified in figure 7.2, which displays the cumulative distribution function of outbound traffic across network prefixes announced by BGP neighbours. Over five years, traffic to approximately 340,000 unique prefixes was observed. Invariably however, an increasing amount is sent to a small group of prefixes – by 2011, over 50% of traffic went to the top 100 prefixes alone. As detailed in chapter 5, this reflects ongoing structural changes in the Internet architecture as content providers interconnect directly edge, *eyeball* networks, and content becomes increasingly consolidated across a set of large content providers and national and regional ISPs.

PREFLEX hinged on the fact that multipath routing state could be maintained and managed in a scalable manner, and these results demonstrate that multipath routing state can be significantly reduced by covering fewer destinations while still benefiting most traffic. Within the MAWI dataset virtually all inbound and outbound traffic could be mapped to 10,000 unique network prefixes. Existing tools such as RouteFlow [RNS+12] are already capable of overlaying routing on commodity switches, but the incurred overhead can still be a concern for production networks. Rather than address the scalability challenges inherent to multipath routing directly, these results suggest that a tangible deployment path lies instead in reducing the scope over which it is applied.

7.2 OpenFlow background

A valuable development in assisting traffic management has been the emergence of SDN, which can facilitate vastly improved flexibility for scalable, policy-based network control. Software defined networks decouple the data plane and control plane, allowing both to evolve independently. A traditional instantiation of a software defined network for data centres is shown in figure 7.3a. Each physical host



(a) (b)

Figure 7.3: OpenFlow (a) architecture and (b) flow entry structure.

runs a number of virtual machines, each connected locally through a edge, software-based switch (such as Open vSwitch [PPK+09]) running on the underlying physical host operating system, which will be denoted as *dom0*. This switch is in turn connected to further forwarding devices, ensuring access to a wider network. The forwarding logic of each device can be accessed and configured by a controller through the establishment of a control channel through a common protocol, of which OpenFlow is the most widely used [MAB+08]. Both software and physical switches are indistinguishable from a controller's perspective: *how* a device implements OpenFlow is immaterial, so long as a forwarding device conforms to the given Application Programming Interface (API).

An OpenFlow flow table is composed of multiple flow entries, shown in figure 7.3b. Each flow entry is comprised of a pattern to be matched, and the corresponding *instructions* to be executed. The *match fields* over which an entry can be compared span from data link to transport layers, covering not only source and destination addresses at each protocol header, but also traffic classes and labels for Virtual LAN (VLAN), MPLS and IPv4 headers. Additionally, a *counter* keeps track of the number of times the entry is matched. If more than one matching entry is found, only the entry with the highest *priority* is processed. Finally, each entry has a pair of associated *timeout* values: a soft timeout, within which an entry is expired if no matching packet arrives, and a hard timeout, by which an entry is irrevocably expired. If neither timeout is set, a flow entry persists indefinitely.

An OpenFlow switch in turn maintains multiple flow tables. Every packet received at an OpenFlow compliant switch is processed along a pipeline which starts by matching the packet against *table 0*. From this first, default table, corresponding *instructions* may redirect the packet for further matching against another table, thereby chaining processing. This pipeline processing ceases once a matching entry fails to include a redirection request, with the accumulated instruction set being executed. In addition to redirections, valid instructions include modifying packet fields, pushing and popping packet tags, and defining through which ports a packet should be forwarded. If at any point no matching entry is found, the packet is buffered at the switch, and the truncated packet header is sent to the controller. Based on the header contents, a controller may decide to install a new flow entry on the switch, or allow the packet to be dropped altogether. Compared to the underlying abstracted network elements which compose the

data path, the controller is often expected to be entirely software based, and as such is not constrained in how it should process packets. In practice, this freedom is curbed as increasing complexity at the controller both reduces the rate at which packets are processed, as well as increasing latency for packets buffered at the switch.

SDN provides an abstraction over which different architectural paradigms can be adapted and even coexist. It does not however prescribe or advocate a specific design – network practitioners must still consider system properties when grappling with fundamental trade-offs affecting consistency, isolation, reliability and efficiency. Some of the design considerations for scalable traffic management were previously described in section 7.1. The overall performance of the described architecture is subject to two further critical trade-offs. Firstly, the granularity at which flow entries are installed determines how often a controller is called to intervene. While installing an entry at a flow granularity may allow fine-grained control of resources, it increases both the load on the controller and the latency of the withheld packet. Conversely, as the granularity becomes coarser, the overhead incurred by the controller is reduced at the cost of flexibility in controlling traffic. Secondly, controller placement is critical [HSM12]. At one extreme, a fully centralized controller is omniscient within a domain at the expense of reliability and scalability. At the other, a distributed system of controllers forsakes consistency and liveness in order to scale robustly.

7.3 Architecture

This section describes INFLEX, an architecture which provides edge domains with greater end-to-end resilience. Rather than probing paths through active or passive means, the network delegates the responsibility for fault detection to end-hosts. The system relies on packet marking at the host to select a path through the local domain. This provides far greater scalability in terms of the proportion of traffic and destinations which can be covered, at the cost of requiring small changes to the end-host TCP/IP stack. INFLEX is therefore particularly suited for managed environments, such as datacenters or enterprise networks, which not only have greater control over the end-host operating system, but also generate large volumes of traffic towards destinations which cannot be readily upgraded.

An overview of the proposed architecture as applied to a single domain is shown in figure 7.4. Hosts are connected to the local network through an OpenFlow-enabled *edge switch*. While edge switches typically reside within each physical machine, alternative aggregation levels such as the top of rack or end of row may also be used. Each switch is configured by a specialized controller which resides locally, referred to as an *inflector*. The local network is configured by a centralized routing controller to provide multiple *virtual routing planes*. While these planes are necessarily intradomain in scope, some degree of interdomain diversity can also be achieved by changing egress node.

The core of the architecture relies on repurposing the DS field in each IP packet to provide an in-band signalling channel between the end-host and the inflector. The header on inbound traffic is set by the edge switch and read by the host, and is used by the inflector to signal which plane a flow has been assigned to. The header on outbound traffic is set by the host and read by the edge switch, and is used by the transport protocol to ensure that all traffic for the flow is forwarded along the given plane. Hosts

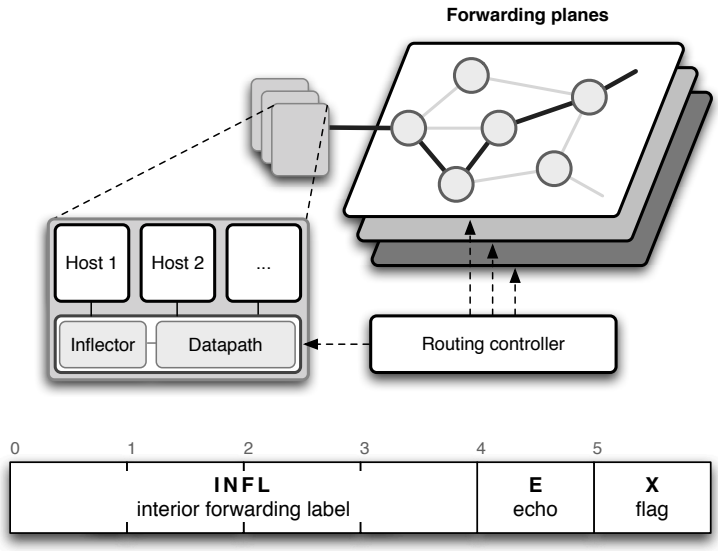


Figure 7.4: INFLEX architecture (above) and header (below). The edge switch forwards traffic across virtual planes set up by a centralized routing service.

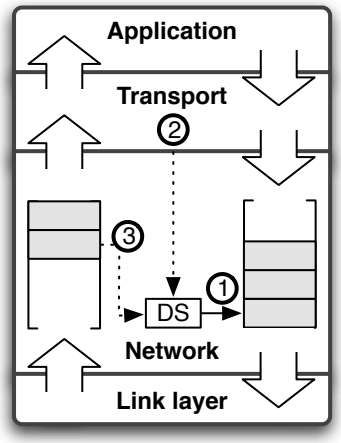
can request a new plane to be assigned by the inflector in case of an end-to-end path fault; this provides efficient cross-layer failure recovery. The DS standard [BBC+98] reserves a pool of code points for local use identified by setting the right-most bit, henceforth referred to as the INFLEX $flag$. When set, the rest of the DS field should be interpreted as containing two fields, shown in figure 7.4. An Interior Forwarding (INF) $label$, which determines the plane over which a packet is forwarded, and an $echo$ bit, which explicitly signals a request from the host or a reply from the network. The remainder of the description of INFLEX is split across its main components: the end-hosts, the edge switch and the inflector.

7.3.1 INFLEX end-hosts

INFLEX hosts set the INF label of outbound packets according to the value assigned by the inflector, reproducing the path re-feedback design pattern introduced in chapter 3. INFLEX however cannot rely on marking by the remote endpoint to trigger network action, as this has been shown to be essentially undeployable. Instead, path requests are initiated by the sender, which must then await for a network reply piggybacked on a returning packet.

The changes required to support this at the sender side network stack are minimal, and are illustrated in figure 7.5a. Every transport connection occurs over a socket, a local structure containing the variables associated to the ongoing flow. At the network layer, the local socket has a DS value which is copied to every outbound packet (point 1). Within INFLEX, the transport protocol can trigger a request (point 2), which leads to a network response contained in incoming packets (point 3).

Assume a transport protocol wishes to switch the plane it is currently assigned. With INFLEX, it can send an *inflection request* by setting the $echo$ bit of the local DS field (point 2, figure 7.5a). All subsequent outbound packets will be marked with the resulting value. The network layer then proceeds



(a)

```
1  ...
2  if (is_inflex(pkt)) {
3    if (!is_inflex(sock) ||
4        (is_pending(sock) && is_reply(pkt))) {
5      copy_label(sock, pkt);
6      clear_echo(sock);
7    }
8
9  } else if (is_inflex(sock))
10   clear_inflex(sock)
11 ...
```

(b)

Figure 7.5: INFLEX (a) host stack and (b) pseudo-code for packet reception.

to inspect inbound packets, waiting for a network response, as delineated in figure 7.5b. After demuxing an incoming packet, pkt, to the corresponding socket, $sock$, a receiver first verifies whether the INFLEX flag is set on the incoming packet (line 2), establishing whether the underlying network supports INFLEX for the given connection. The receiver must then decide whether it should change the virtual plane the socket is currently assigned. This can only happen under two conditions. Firstly, if the DS value for the current socket does not have the INFLEX flag set (line 3). This typically occurs on flow start, where a connection is spawned with a default DS value. Secondly, if the local DS value has the echo bit set, there is a *pending* inflection request. If the incoming packet has the same bit set, it corresponds to the network *reply* (line 4). Under both previous cases, the connection switches forwarding plane by copy the interior forwarding label from the incoming packet to the local socket, and setting the INFLEX flag (lines 5-6). These changes are all applied at the IP layer – transport protocols need only to decide when to send inflection requests – while applications can remain unchanged.

7.3.2 The edge switch

The edge switch is primarily responsible for mapping INFLEX marked packets to the appropriate forwarding plane. On start up its datapath is configured by the local *inflector*, which installs the appropriate flow entries on it in order to construct the processing pipeline in figure 7.6. This pipeline can be partitioned into three distinct blocks, responsible for *triaging*, *policing* and *inflecting* packets. For clarity, the processing pipeline is conceptually described as a sequence of flow matches across distinct tables. In practice, an implementer is free to collapse flow tables and entries to improve performance. An important safeguard is that a legacy pipeline must be present, establishing a default forwarding plane expected to be used by traffic to which INFLEX is not applicable.

The *triage* phase is responsible for distinguishing whether a packet is *capable* of using INFLEX. Firstly, INFLEX is only applicable to IP packets. Traffic is then differentiated according to the port on which the packet arrived: if connected to a host, the interface is said to be *internal*, otherwise it is

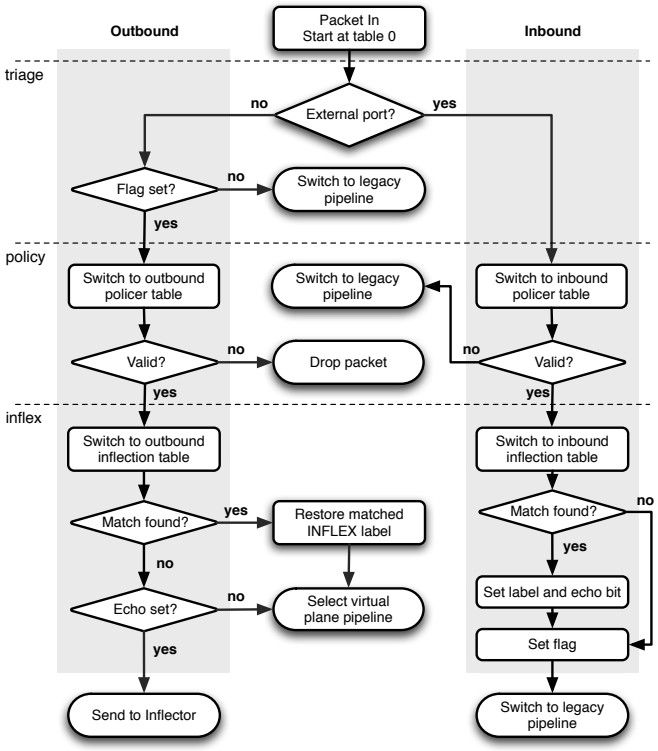


Figure 7.6: Pipeline installed to the edge switch datapath.

external. Any inbound IP traffic may potentially be INFLEX capable and as such can proceed to the next stage. For outbound IP traffic, only packets with the INFLEX flag set require further processing. Packets for which this flag is not set are assumed to be legacy traffic.

The *policy* phase decides whether a packet is *permitted* to use INFLEX. For either direction, a packet is compared against a policer table, which contains a set of rules describing local policy concerning INFLEX usage. The rules applied to each direction however may differ, particularly since outbound packets can be further scrutinized according to the INF $label$. For example, this allows the outbound policer to enforce which virtual planes are available to specific tenants or applications. For this reason, the action applied if a packet is matched within the policer table also differs according to direction. For inbound traffic, a matching rule indicates that the packet does not satisfy local requirements for INFLEX use, and is consequently treated as legacy traffic. For outbound traffic, a packet is already marked as being INFLEX capable. Any matching entry therefore indicates that it is in violation of local policy and should consequently be dropped.

Finally, the *inflex* phase processes the respective header and forwards the packet. A packet is first matched against an *inflection* table in either direction. This table is detailed in the next section, and can be assumed to contain no matching entry initially. For outbound traffic, the packet is typically redirected to the plane mapped by the interior forwarding label. The one exception are inflection requests, which are forwarded to the local inflector for further processing. For inbound traffic, the INFLEX flag is marked in order to notify hosts that the flow is INFLEX capable, and the packet is then processed according to

the legacy pipeline.

7.3.3 The inflector

Each edge switch is controlled by an inflector, an SDN controller expected to reside locally. An inflector is firstly responsible for configuring the underlying datapath according to the previously described pipeline. Secondly, an inflector must process inflection requests.

Inflection requests require network intervention in assigning a packet to a forwarding plane. The dynamic nature of this decision process cannot readily be instantiated as a set of static rules at the edge switch, since a same flow must be able to be reassigned to a different plane in case of path faults. Therefore, inflection requests intercepted at the edge switch must be sent to a controller for further processing. Rather than overloading a centralized controller however, this decision can be taken locally – since the inflector manages the local rules associated to each virtual network, it already has full knowledge of the routing table associated to each plane. Upon receiving such a request, the inflector proceeds in three steps. It first verifies which virtual networks maintain a valid route for the given destination address. Given this list of potential planes, it then inspects local policy to verify which planes the packet is allowed to use. The intercepted packet contains the plane which the flow is currently using – this plane should be excluded from the candidate list unless there is no other option available. Finally, a single plane, identified by an interior forwarding label, is selected from the resulting list of candidates. The selection algorithm is not prescribed by the INFLEX specification, but a reasonable baseline is to select a routing entry proportionally to the assigned route weight.

Having selected an appropriate plane, the inflector installs forwarding rules into either *inflection table*. In the inbound direction, all packets matching the reverse flow are set to be marked with the corresponding INF $label$. This conveys the selected forwarding plane back to the host. In the outbound direction, all packets matching the flow are to be processed according to the $label$. This guarantees that any packet sent between the inflection request and its response are forwarded in a consistent manner. Rules installed to the inflection tables are ephemeral by nature, with a hard timeout of 1 second (the minimum permitted in the OpenFlow standard). This enables per-flow granularity with minimum flow state while also rate limiting inflection requests. Furthermore, flow entries can be configured to be sent to the controller upon expiry. This potentially allows the inflector to collect realtime information on the availability of each forwarding plane, allowing for further refinement of the plane selection algorithm.

7.4 Analysis

This section details the evaluation of INFLEX as well as details pertaining to its implementation. A reference implementation of the inflector was developed as a component of the POX network controller [pox]. Additionally, INFLEX support for TCP was added to the Linux kernel, and is available as a small patch for version 3.8. All source code, as well as a virtual machine to replicate subsequent tests, is being made publicly available. The use of POX in particular invalidates any rigorous performance evaluation, as the implementation is not geared towards efficiency. Instead, the contributed code acts as a proof-of-concept for INFLEX, allowing the mechanics of the specification to be inspected and fine-tuned.

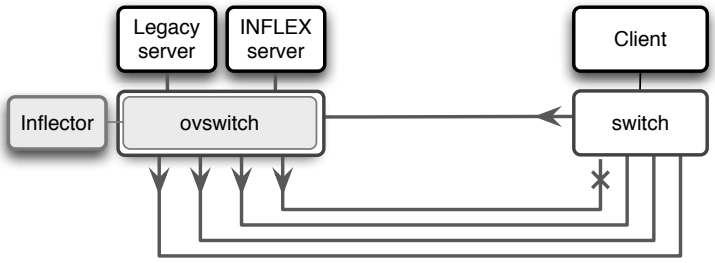


Figure 7.7: Simulation setup.

To this end, a simple evaluation scenario is used, illustrated in figure 7.7. On one end is an INFLEX capable domain: a set of virtual hosts acting as *servers* connected to an Open vSwitch edge switch controlled by an inflector. On the other end is a remote *client*. Typically this is an end-user device outside network operator control. We assume that the client is running a legacy network stack and connected to a switch with no SDN functionality. A single physical connection between the client and this switch acts as the bottleneck for all flows, with the bandwidth set to 10Mb/s. The edge switch has four potential planes over which it can forward traffic between the *servers* and the *client*. We simulate failures within the INFLEX domain by artificially dropping all forwarded packets belonging to a given plane; we denote this plane as being *down*. At any given moment one of the four available planes is down; each such simulated failure lasts for 15 seconds at a time, affecting planes cyclically. The reverse path, connecting from client to server, is always assumed to be functional. Propagation delay between both switches is set to 50ms, resulting in a base round trip time between servers and client of 100ms.

7.4.1 Sender-side resilience

The first case study under review is one of the most common use cases for datacenters: a remote client *downloading* data from hosted servers. Under the conditions described previously, the forwarding path will be periodically affected by recurring failures. Since the nature and the origin of the fault are not always apparent to network devices, it is assumed that network elements within the INFLEX domain have no indication of the ongoing failure. Instead, it is up to the servers to detect and recover from perceived problems by issuing inflection requests. Clearly, requesting a path incurs some cost to network and host alike. For the network, an inflection request requires additional processing. For the host, this processing manifests itself as increased delay. This begs the question: when should a host request an inflection? The obvious candidate is to piggyback inflection requests on retransmissions spawned by retransmission timeout (RTO). This leverages an existing transport mechanism which is well understood and only triggered under anomalous path conditions (as opposed to congestive losses). From the perspective of the host, any delay incurred by the inflection request is amortized by the retransmission timeout itself, which has a minimum value of 1 second. From the perspective of the network, such inflection requests should be both rare, reducing the amount of processing required, and critical to improve availability, justifying the expense in addressing them.

Figure 7.8 displays the congestion window over time for two concurrent flows towards a remote

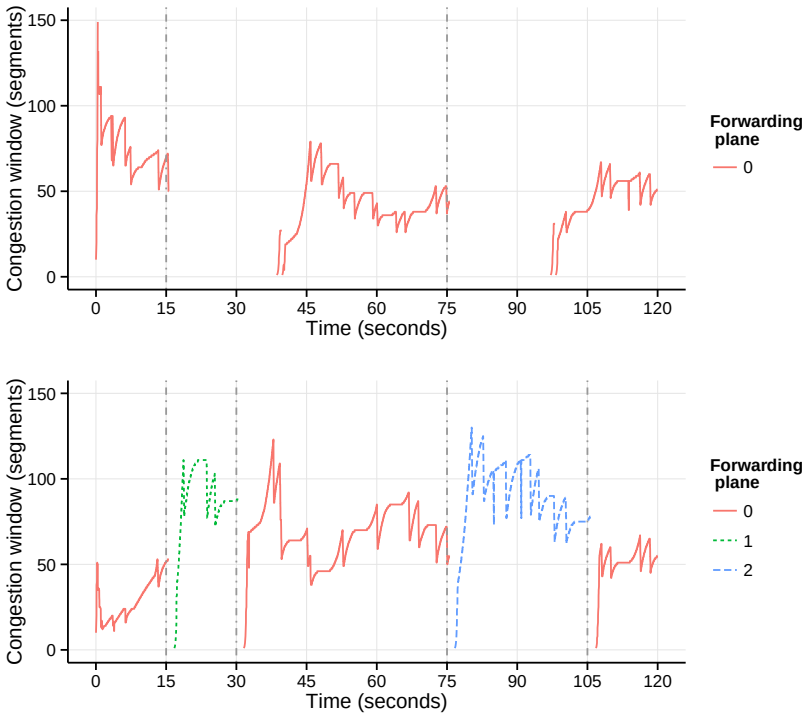


Figure 7.8: Congestion window for concurrent downloads towards client from legacy (above) and INFLEX (below) servers.

client. The first connection traced is a download from a server without INFLEX support, in which all packets are forwarded over the default path. The vertical lines signal the points at which the default forwarding path, plane 0, fails. Despite only failing for 15sec, the disruption to the transport flow lasts twice as long due to the exponential nature of the retransmission timeout, which doubles in duration at each occurrence. The second connection traced is a download occurring in parallel from an INFLEX capable server. In this case, each path failure is recovered by sending an inflection request on each retransmission timeout. The returned path is randomly assigned, as our basic proof-of-concept inflector does not currently keep track of network conditions. The time between path failure and flow recovery is directly tied to the RTO, in this case taking approximately one second. This value cannot be improved upon within the INFLEX framework, as the duration of flow entries introduced by inflection requests has a minimum timeout of 1 second. Conveniently however, this matches the lower bound of the RTO as defined by TCP, and it is therefore unlikely that a transport protocol would desire faster fail-over. In practice, the recovery time may be extended in order to account for spurious timeouts. For connections over wireless media in particular, timeouts may occur due to transient effects such as interference. While this is functionally equivalent to path failure, the transient nature of such events does not always require changes to the forwarding path.

An interesting implication of figure 7.8 is that **TCP senders using INFLEX can accommodate path fail-over seamlessly**. Retransmissions conveniently distinguish between congestion events, triggering fast retransmission and similar heuristics, and pathological network conditions, which spawn

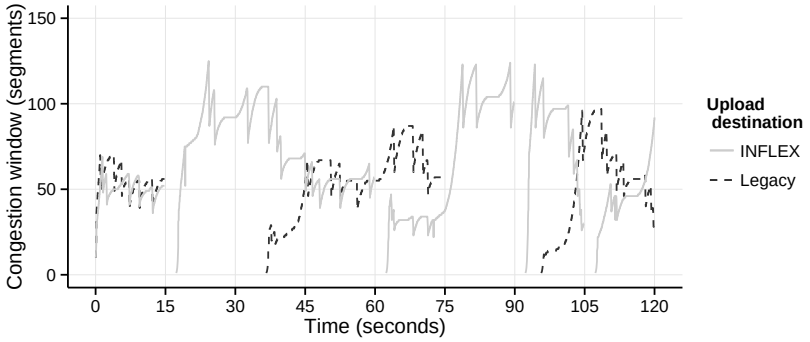


Figure 7.9: Congestion window for concurrent uploads from client towards servers.

repeated retransmission timeouts. In the case of the latter, the adopted response is to reset the congestion window and resume slow start – effectively restarting the flow. This behaviour is extremely conservative, but is a natural corollary of assuming as little as possible about the underlying network. As a result, no substantial change is required to the congestion control mechanisms employed by TCP in retrofitting cross-layer path fail-over at the sender using INFLEX.

7.4.2 Receiver-side resilience

Path failures can also affect the reverse path with equally nefarious consequences: the sender will repeatedly timeout in the absence of acknowledgements from the receiver. Unlike failures affecting the forward path however, the INFLEX host does not actively track the reliability of the ongoing connection. TCP is sender driven, with receivers primarily generating acknowledgements as a response to inbound data. Hence, the reverse path lacks the reliable delivery mechanisms available in the forward path; if the *TCP Timestamp* option is not used, the receiver often lacks even an accurate RTT estimate. Furthermore, in the absence of data packets to be sent, there is no RTO on which to trigger inflection requests.

A receiver must instead rely on inferring path failure from the packet inter-arrival time when generating duplicate acknowledgements. With the exception of cases where immediate receiver feedback is required, such as a TCP timestamp request, duplicate acknowledgements are typically sent on the arrival of out-of-order data. Under path failure, the arrival time between such out-of-order events will rise exponentially as the sender TCP stack becomes tied to its own retransmission timeout. This behaviour is illustrated in figures 7.9 and 7.10, which show the result of using INFLEX with the same experimental setup but a reversed flow of data. Figure 7.9 displays the evolution of the congestion window size over time as the client uploads data concurrently to both a legacy and an INFLEX server. While the single forwarding path does not experience outages, the reverse path is periodically affected by failures. The corresponding data packet inter-arrival time is shown in figure 7.10, with each sample point also displaying the routing plane used. For an ongoing TCP flow with sufficient data from the application layer the packet inter-arrival time at the receiver should be consistently low. RTT level dynamics are apparent on slow start, in which the sender is clocked by incoming ACKs, and during congestion events, in which out-of-order delivery temporarily affects the throughput. On path failure however, the inter-arrival time increases exponentially, with each inbound packet triggering a duplicate acknowledgement. For the

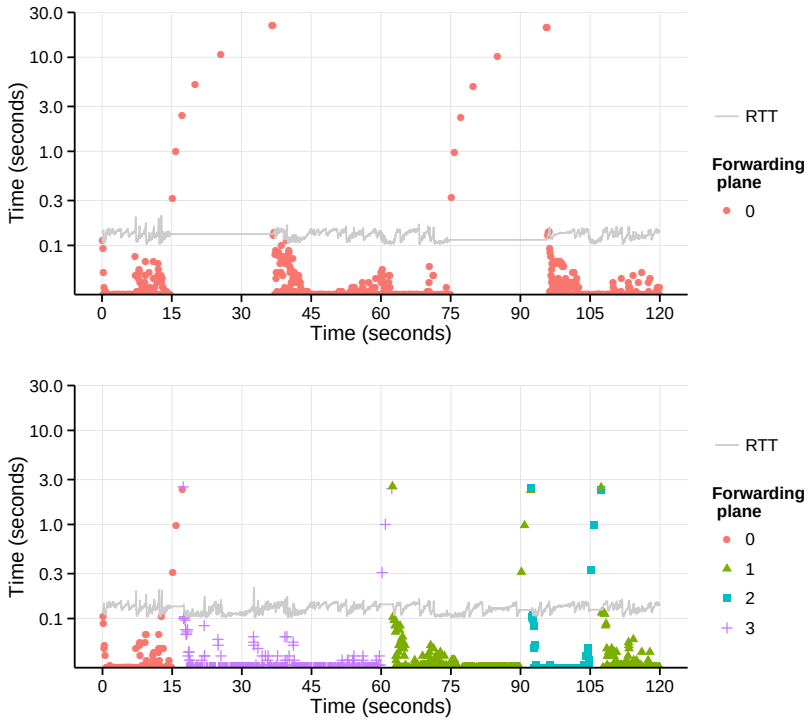


Figure 7.10: Data packet inter-arrival time for legacy (above) and INFLEX (below) receivers.

upload to the legacy server, successive RTOs result in a recovery time of nearly 30sec.

An INFLEX receiver can use this information to decide when to trigger an inflection request. It can achieve this by setting a threshold for the time elapsed between duplicate acknowledgements, henceforth referred to as $dupthresh$. Comparatively to the sender, the receiver should be more conservative, as by design it has less information on which to act upon and does not typically exert control on the congestive feedback loop. Furthermore, neither sender nor receiver can reliably detect whether the forward or reverse path are at fault. By acting conservatively, a receiver allows the sender, which may also be INFLEX capable, to initiate recovery before trying to correct the reverse path. For the experiment displayed in figure 7.10, the $dupthresh$ is set to twice the RTO, resulting in an overall downtime of approximately 3 seconds. Since each data point is generated on inbound data packets, recovery is signalled by a packet pair. A first inbound packet exceeding $dupthresh$ triggers an inflection request, which piggybacks on the acknowledgement sent out in response. A second inbound packet returns approximately 1 RTT later with the forwarding plane assigned by the network attached. Clearly some failures may not be recoverable, particularly if the remote host is not INFLEX capable and the fault lies on the reverse path. Nonetheless, the overhead incurred at the host is negligible, merely complementing congestion avoidance mechanisms with additional signalling. Remarkably, INFLEX incurs no additional memory costs at the host, operating as an extended API over the existing $inet\_connection$ socket, rendering it equally applicable to all transport protocols which use this socket structure, such as SCTP and Datagram Congestion Control Protocol (DCCP).

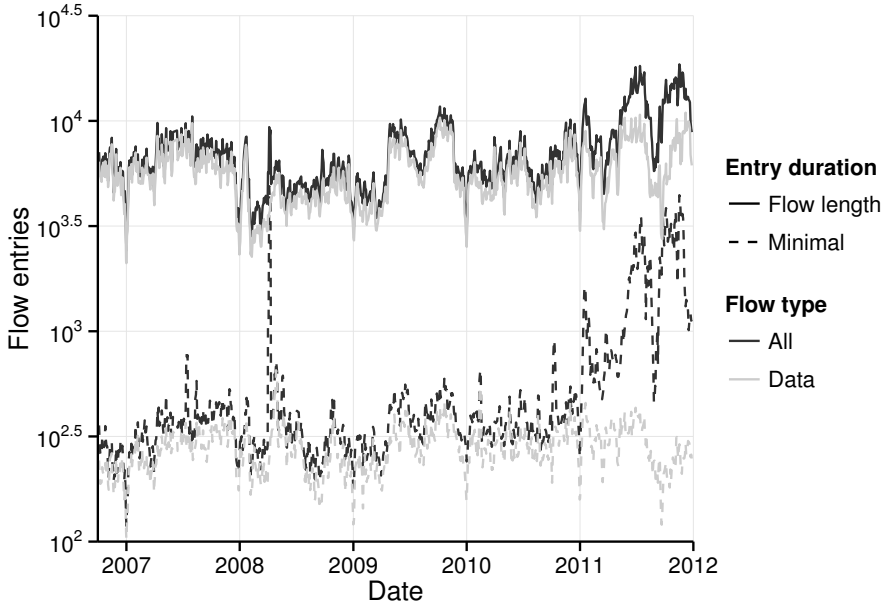


Figure 7.11: Mean flow state for outbound traffic.

7.4.3 Network overhead

The granularity at which an SDN deployment should manage traffic is often subject to debate. On one hand, hardware advances such as Ternary Content-addressable Memories (TCAMs) offer fast lookup times over large tables, affording flow precision for many potential SDN deployments. On the other, deployments will often include cheaper, more flexible software switches which are less capable of scaling performance with the number of flow entries. Importantly, operating on a per-flow granularity is more likely to overload the controller, which itself can be a considerable source of latency. As a result, managing flow aggregates is often the preferred means of reducing this overhead, at the cost of flexibility in affecting flows individually.

INFLEX does neither strictly, exerting network control at a sub-flow granularity while pushing flow state to the end-host. Figure 7.11 investigates the relative expected overhead incurred by the network on adopting such an architecture. The graph tracks the mean flow state from applying different flow entry policies for outbound traffic in the MAWI dataset. The solid lines track the resulting flow table size if traditional per-flow state were maintained, with every unique five tuple inserting a table entry for the entirety of the flow's lifetime. This is equivalent to the mean number of flows at the observed link and is further refined according to whether data was traced for the unique five tuple. For domains which exchange traffic with the wider Internet, per-flow state can be particularly crippling as malicious SYN floods and port scans regularly inflate the required state in the network. Such attacks had visible impact in 2011 in particular, nearly doubling the number of flows.

INFLEX however inserts ephemeral rules in response to inflection requests. For the worst possible case, all existing flows would trigger an inflection request simultaneously – matching the overhead incurred by a per-flow approach. In practice even this is overly pessimistic, as an inflector could resort

to a per-aggregate granularity in the case of widespread outages. Actual network state would strongly depend on the exact inflection strategy adopted by the transport protocol. One practical reference point is to investigate the resulting overhead if paths were requested on flow start, as this number will exceed retransmission timeouts under normal operating conditions. This is further illustrated in figure 7.11, which also tracks flow table size if each unique five tuple were to only generate a flow entry for 1 second, the minimum expiry time for OpenFlow. This is functionally equivalent to the flow arrival rate, and determines the expected number of requests per second sent to the controller. The resulting flow table size is reduced dramatically in comparison to the traditional case where state is allocated for the duration of the flow, and the order of magnitude difference is crucial for software switches in particular. However, under such conditions state becomes more strained by the large fluctuations imposed by DoS attacks, suggesting that inflection requests should only be used after connection establishment; this corresponds to the grey dotted line in figure 7.11. Importantly, such an approach also opens the possibility of using inflection requests for assisting traffic management in addition to enabling improved resilience.

7.5 Unifying approaches

In chapter 3, PREFLEX was presented primarily as an architecture for dynamic traffic management, but one in which resilience was identified as a potential benefit given sufficiently adaptive control. In this chapter, the opposite approach was taken: deriving a traffic management solution from an architecture first built around resilience.

INFLEX corrected two potential shortcomings in PREFLEX. Firstly, by eschewing the need for receiver collaboration, INFLEX is unilaterally deployable by edge domains. Secondly, path acquisition was decoupled from whether a host had timely information on path quality, removing the requirement for flows to incur additional delay on flow start. As a side-effect, by not tying network processing to SYN packets, INFLEX is also less susceptible to DoS attacks. Nonetheless, INFLEX as presented thus far is geared solely towards resilience. The remainder of this section details how it can be extended to provide the benefits offered by PREFLEX, which are typically of direct interest to operators in particular.

7.5.1 Traffic management

PREFLEX offered a fine-grained mechanism for traffic engineering, allowing networks to assign flowlets to network paths. INFLEX on the other hand only inflects flows upon request. This seemingly reduces the opportunities for traffic balancing to when path faults arise. The host modifications introduced in section 7.3.1 however make a provision for flow balancing: on flow start, an INFLEX capable flow has no label associated to the socket, and must consequently acquire whatever label is provided by the edge switch.

Operators can therefore assign flows to specific paths by marking inbound packets with the appropriate label. Clearly, only the first packet of a flow will take effect – once a flow has acquired an INF label, it will no longer inspect subsequent INFLEX headers unless an inflection request is pending. As such, one possibility for operators is to intercept the inbound SYN packet and tag it with the appropriate label, in much the same manner as PREFLEX. This presents two challenges. Firstly, it introduces a DoS

vulnerability as the amount of network processing is tied to the volume of inbound SYN packets, which can greatly exceed the number of actual flows as shown in figure 7.11. Secondly, there is no obvious manner of intercepting SYN packets using OpenFlow as TCP flags are not actionable packet header fields. This means that such functionality would either have to be implemented by specific middleboxes, adding complexity to the proposed architecture, or flow table entries would have to span the entirety of a flow, increasing table size significantly. In either case, pushing packets to the controller on flow start would necessarily incur a performance penalty for all flows.

A faster, more scalable approach relies on the fact that hosts will ignore the INFLEX header once a label has been acquired. As such, marking a stream of packets within a flow will have no negative repercussions. This means that rather than redirecting specific packets to the controller for marking, a rule for marking the inflection label can be installed a priori to the datapath. Within the flowchart presented in figure 7.6, this corresponds to installing a wildcarded entry on the inbound inflection table. Since such an entry would have a lower priority than pending inflection requests, the basic INFLEX mechanism described in section 7.3 remains unaffected. Since the rule can be wildcarded by network prefix , the number of rules would likely be small. Figure 7.2 suggests that over 50% of traffic is destined to as few as 100 prefixes. Interestingly, such marking rules need not be applied by the edge switch, instead being performed at a hardware switch relying on TCAM. This both allows larger tables and reduces lookup times, while still delegating policy enforcement to edge switches.

A significant open issue that remains in this regard is how best to dynamically assign flows to routes, as OpenFlow does not support probabilistic behaviour being pushed to the datapath. The most obvious approach is to change the mapping between flow and route periodically, in a proportional manner to the desired split. This raises two potential concerns. Firstly, flow entries have a time granularity of one second. While an equal split amongst two paths can be achieved by alternating between flow entries every second, finer flow split ratios over a greater number of paths will incur a much higher periodicity. Secondly, the impact of this periodicity on overall system behaviour is not clear. A high path switching frequency may degrade the performance of the switch due to rule churn. A low frequency on the other hand may induce bursty traffic allocation over short time scales, particularly if flow arrival rates are irregular or the bulk of traffic is composed by short flows.

Under some conditions, a network may need to reassign a flow to another path proactively. Operators may desire to assign particularly large flows to specific links, clear a link for programmed downtime or allow routing changes to converge before actually using new routes. In any case, INFLEX provides operators the means to force a path inflection. This can be achieved in two steps. The network first clears the INFLEX header for an inbound packet. This signals to the end-host that the network is no longer INFLEX capable, leading the network stack to clear the existing DS field, as detailed in figure 7.5b. Next, a network sets the INFLEX header for the subsequent packet in the flow. This leads the receiver to once more adopt INFLEX and acquire the label. While such forceful behaviour can disrupt transport, it can prove valuable, and networks already have the means to achieve such goals through the cumbersome manipulation of routing. In this case, INFLEX merely provides existing functionality more flexibly for

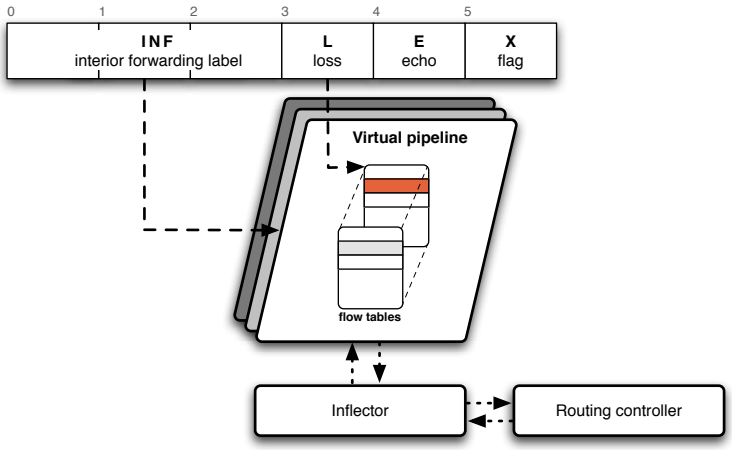


Figure 7.12: Extending INFLEX for congestion balancing. The INF label on outbound packets is used to select the virtual pipeline, while the $loss$ bit is used to toggle between flow tables.

the network and more transparently for the end-host.

7.5.2 Congestion management

One of the unique propositions of PREFLEX was to enable networks to perform traffic engineering according to congestion rather than just link load. This congestion balancing is particularly useful when there is an asymmetry in available bandwidth between providers and when most traffic is not multipath capable. In order to allow the use of congestion balancing as detailed in chapter 4, INFLEX requires two modifications: hosts must *signal* loss to the network, and the network must *account* for lost packets.

Signalling loss requires marking the outbound INFLEX header. Given the constraints imposed by the DS field, the only option is to reduce the size of the INF label to three bits, freeing one bit in the process. At the transport layer, end-hosts would subsequently set this bit on all retransmissions. This still allows for seven potential paths per destination, in addition to the default forwarding path.

Figure 7.12 displays the resulting header, alongside the changes internal to the network for counting lost packets. At the edge switch, the datapath remains unchanged, with the INF label still being used to select the appropriate virtual pipeline for the outbound packet. Unfortunately, OpenFlow does not provide more than one counter per table entry. Given this constraint, extracting the necessary metrics for performing congestion balancing proceeds by maintaining two replicas of the appropriate routing table. One flow table is responsible for keeping count of the number of retransmitted packets per entry, while the other accounts for all other packets. While slightly contrived, this implementation is sufficient for providing congestion balancing as described in chapter 4, and much of the underlying details can be abstracted from the routing controller.

The inflector has the responsibility of exposing an extended OpenFlow API which both implements the routing policies dictated centrally by the routing controller, and returns statistics necessary for route computation. Whenever a route is *installed* to a given plane, an inflector must insert the corresponding rule into the two parallel flow tables for loss accountability. Whenever a central controller requests

counter metrics for a given entry, the inflector is responsible for returning not only the total number of packets observed for a given route, but also the number of retransmissions. This cleanly decouples the routing controller from the underlying implementation of INFLEX at the edge switch.

7.6 Conclusions

This chapter presented INFLEX, a scalable and easily deployable end-to-end resilience framework based on the cross-layer control of an SDN-enabled network layer. The proposed architecture is shown to perform end-to-end path fail-over on much shorter time scales than existing solutions and is inherently modular, providing failure recovery through cooperation between end-hosts and the IP network. In comparison to reliability mechanisms operating purely at the transport layer, such as MPTCP or SCTP, INFLEX enables transport resilience when communicating with legacy endpoints and does not require host multi-homing. Conversely, when compared to mechanisms operating purely at the network layer, INFLEX provides end-to-end visibility into path failures, allowing both fast detection and the implementation of remedial actions requiring fine-grained network control. The architecture design presented is based on network measurements and implemented as a set of extensions to the Linux kernel and a popular OpenFlow controller. This implementation is then evaluated experimentally, demonstrating that high availability over multiple routing planes can be achieved without compromising network control scalability.

Chapter 8

Conclusions

The presented body of work identifies opportunities and explores strategies for resource pooling through the application of *re-feedback*. Resource pooling has evolved to being performed by different stakeholders unilaterally: end-hosts, network operators and content providers all attempt to pool traffic by differing means, and in a potentially conflicting manner. While recent work [KV05, WHB08, WRGH11] has lent credence to pushing resource pooling towards the edge, this ultimately ignores the tussle over how traffic is managed between these stakeholders. Network operators attempt to exploit path diversity through a combination of route optimization and traffic balancing. Collectively, these traffic engineering methods assist networks in minimizing the costs incurred by shifting traffic. Conversely, hosts are increasingly capable of pooling traffic across paths either through the use of overlay networks or multipath congestion control, neither of which necessarily share the objectives of the underlying network.

The ultimate end-product of this thesis is INFLEX, a cross-layer architecture for resource pooling. The proposed traffic management solution is based on extensive measurement data, easily deployable, and elegant, incurring no significant overhead at end-hosts and requiring limited, scalable processing within the network. Importantly, it addresses concerns which are relevant to operators and users alike, and is shown to be:

Efficient through the application of congestion balancing. A model is derived for traffic assignment across different paths in proportion to the amount of congestion incurred by ongoing flows. In contrast to past efforts, this allows the network to balance traffic according to end-to-end loss. The resulting system is shown to make better use of available end-to-end capacity than existing methods traditionally applied by operators for traffic engineering purposes.

Flexible through the use of cross-layer signalling. *Path re-feedback* in particular allows varying degrees of control by the network over how traffic is assigned to links while minimizing required state. Likewise, hosts are afforded path diversity but are free to opt out at no cost. This contrasts positively with existing solutions such as MPTCP, where path setup can incur additional bandwidth and latency.

Robust through the delegation of fault detection to hosts. The transport protocol can request path changes on-demand, which explicitly signals end-to-end path faults to the network. This informa-

tion can in turn be crowd-sourced from ongoing flows by the network and assist in fault location and reparation.

8.1 Summary of contributions

This section summarises the contributions of this thesis in light of the original problem statement:

> *Given the nature of Internet traffic, how can the current architecture be realigned to facilitate resource pooling at both network and transport layers?*

8.1.1 Internet traffic characterisation

This thesis began by providing a broader context within which to frame the evolution of resource pooling. Chapter 2 traces how successive waves of new stakeholders and novel applications have influenced and shaped the protocols and tools which form the Internet. Importantly, this chapter highlights traffic management as an architectural afterthought, best understood as being driven by the nature of the traffic and available capacity at hand.

In order to inform the design of a resource pooling architecture, an extensive longitudinal study of Internet interdomain traffic was undertaken and documented in chapter 5. The analysis of the MAWI dataset characterized over five years of TCP traces in relation to where traffic flows. The resulting longitudinal snapshot, from a single transit link, corroborates previous findings [LIJM+10a] showing a shift in content distribution. Of particular relevance to the present work was the increased consolidation of traffic across a smaller set of stakeholders, with the ten most popular ASes alone accounting for over 50% of all traffic in either direction. From an operator perspective, this allows ample opportunity for balancing traffic by manipulating a smaller set of traffic prefixes.

Section 5.3 further presented a novel RTT recovery mechanism based on cumulative histogram construction and peak detection which assisted in analysing how large-scale shifts in where traffic flows from has impacted end-to-end delay. The results highlight that traffic downstream is moving further away from Japan as content is not only placed closer to consumers and bypasses the transit link entirely, but also moves eastwards within the United States. Conversely, upstream traffic has moved closer as data is predominantly uploaded to the very same co-location centres and content providers from which data is retrieved. Within the observed time frame, the average RTT for inbound and outbound data has converged to approximately $200ms$.

RTT recovery is in turn used to scalably reconstruct and classify flow throughput behaviour in chapter 6. The main contribution of this thread of work is in providing a re-evaluation of commonly held assumptions regarding Internet flow rates. This was done by systematically identifying artificial constraints to TCP traffic throughput across three categories: *application pacing*, *host limiting* and *receiver shaping*. The resulting analysis shows that flow rates are not typically dictated by TCP congestion control alone, and has significant implications on how to reason about resource sharing in particular. The findings equally confirm that TCP throughput is mostly determined by the actions of the sender and that continuing operating system updates have progressively lifted many of the limitations inherent to socket buffer sizes. These changes have allowed smaller flows to increase throughput at a far higher rate

than larger flows, which are more often than not affected by other mechanisms of traffic shaping. This means that, although there is a correlation between flow volume in bytes and throughput, the relationship between the two is non-linear and has changed with time.

8.1.2 Architectural contributions

The problem statement singled out the network and transport layers as the point in the Internet architecture where the tussle surrounding traffic management is greatest. At the hosts, TCP is designed to saturate available bandwidth, making efficient use of the network. Within the network, operators often attempt to minimize the maximal link utilization in order to contain costs and ensure some degree of QoS. Re-feedback was originally proposed by Briscoe et al. [BJCG+05] and applied to the re-ECN protocol [BJMS08], which was put forward as a potential solution for resolving the contention surrounding resource sharing – how capacity is shared end-to-end. It was therefore naturally identified as a starting point for addressing the contention surrounding resource pooling – how traffic is allocated amongst paths.

Chapter 3 began by stripping down re-ECN into a coarser, more practical method for Loss Exposure (LEX). Re-ECN was designed to provide congestion accountability, allowing each domain to precisely quantify the congestion incurred by others. Operators could then be expected to charge their providers according to congestion volume - a metric exposed by re-ECN. Exposing congestion in an accurate and enforceable manner however made an otherwise elegant mechanism complicated to deploy. LEX stripped down the functionality of re-ECN to address a different problem. LEX transmits information on loss as viewed by the transport layer to network resources, allowing traffic engineering to be performed taking into account end-to-end path quality. As such, LEX requires neither congestion marking at bottleneck routers, nor traffic shaping policers at the edges.

The use of loss exposure was then complemented with Path Re-Feedback (PREF), a cross-layer signalling mechanism between the transport and network layer, also presented in chapter 3. PREF offers the ability for the network to select and offer paths to hosts, thereby unlocking the path diversity which already exists at stub domains such as ISPs, CDNs and enterprise networks. Within the self-imposed constraints of IPv4 deployability, the resulting mechanism is necessarily simple, but also shown to be surprisingly versatile. Historically, the PREF field can be interpreted as a synthesis of the previously sanctioned uses for the same header space: neither strictly abiding by the thesis that end-hosts should independently determine their own type of service, nor aligning itself with the antithetical view that the network alone should establish differentiated services.

Finally, chapter 7 refines these proposals in the context of SDN. Section 7.1 grounded the design of INFLEX on the observations made in chapters 5 and 6. In comparison to PREFLEX, INFLEX addresses resilience explicitly and minimizes the occurrence of path switching due to the delay incurred by network processing. Existing transport proposals such as SCTP and MPTCP provide resilience through precautionary establishment of multiple end-to-end paths. For short flows, such set up costs can be prohibitive in terms of latency. By design, INFLEX provides resilience at no additional cost to flows: hosts can simply request to change path in reaction to network events.

8.1.3 Resource pooling enhancements

Given the Internet architecture should not dictate the outcome of the tussle between end-host and network solutions for resource pooling, the proposed architectural additions attempt to make both aware of each other. This enables novel end-to-end traffic management techniques which are not currently possible, requiring only sender-side modifications for immediate deployment.

Chapter 4 describes one possible extreme of the tussle, in which the network is entirely responsible for resource pooling. A congestion balancer is derived in which PREFLEX can reap many of the benefits of MPTCP by balancing flowlets according to loss, but without the need for application changes. Unlike most existing TE methods, congestion balancing with PREFLEX is designed to minimize the impact of route changes on the transport layer and as such is assessed by its impact on transport metrics rather than traffic aggregates. The use of congestion balancing in the network is shown to not only lead to a more efficient use of network capacity, but also a reduction of flow completion times.

Chapter 7 adopted the principles of path re-feedback to provide on-demand path fail-over for IP traffic. Under the current architecture, path failures are largely a network responsibility: operators are expected to detect, repair and recover from faults which may originate from beyond their network domain. Detection often requires scale – the reliability with which a fault can be identified relies on the proportion of traffic affected – and in many cases faults affecting individual flows may go undetected. Reparation varies according to the nature and origin of the fault, and is often not easily automated and itself error-prone. In either case, the amount of time expended in detection and repair can often preclude recovery, since most flows will terminate after successive timeouts.

The presented solution for resilient traffic management, INFLEX, is both unilaterally deployable, providing benefits even when adopted by individual domains, and inherently end-to-end, potentially covering third party failures. Since INFLEX operates as an extension to the network abstraction provided by IP, it can be used by all transport protocols. At the host, the proposed architecture allows transport protocols to switch network paths at a timescale which avoids flow disruption and which can be transparently integrated into existing congestion control mechanisms. Within the network, INFLEX provides both greater insight into end-to-end path quality, assisting fault detection, and more control over flow path assignment, enabling more effective fault recovery. INFLEX was implemented and verified experimentally through modifications to both the TCP/IP network stack and a popular OpenFlow controller [pox] and made publicly available.

8.2 Future work

While chapter 7 ties most of the work of the previous chapters together, some threads remain outstanding. The most immediate concern is that the current implementation of INFLEX validated in section 7.4 does not yet allow for congestion balancing, described in chapter 4. Most of the required changes are accounted for and described in detail in section 7.5. In the longer term, this work has raised several higher-level issues which are outside the scope of this thesis and as such remain potential avenues for future work:

System stability. Concerns over route flapping were raised both in chapter 4 and 7. With congestion balancing the time period between updates of the flow split ratio is paced according to the sparseness of loss, but chapter 4 falls short of providing either a formal proof for stability or extensive enough evaluation. The changes introduced in chapter 7 however are likely to significantly reduce the likelihood of route flapping, notwithstanding the limitations of OpenFlow raised in section 7.5. Since in INFLEX resilience and efficiency are decoupled, an operator can afford to be more conservative in congestion balancing without forsaking responsiveness to failures. Lagging system response in this manner does not affect an operator's primary objective in adopting congestion balancing: namely, keeping upstream providers in check over the quality-of-service they provide and distributing traffic accordingly.

Quality-of-service. Due to its contentious nature, most of the work in this thesis meanders around the subject of QoS. The INFLEX architecture however provides a clear path of deployment for some degree of service differentiation. The policy mechanisms described in section 7.3 allow operators to select which forwarding planes are exposed to individual hosts or flows. Further integration with existing resource management tools within datacenters would allow such policies to be enforced by tenant. Traditional approaches to QoS typically rely on operators provisioning their networks according to preordained service types such as those requiring high throughput or low latency. Within the design principles enunciated in chapter 3, a more reasonable solution would be to apply division of labour in providing QoS. Instead of normalising the expected loss rate over multiple paths as described in chapter 4, an operator could instead adjust congestion balancing to provide differentiated tiers. Hosts would then be responsible in assembling higher quality service from essentially parallel, best-effort forwarding planes according to their needs.

Extending transport. A counter-intuitive result from chapter 6 is that TCP is both becoming both increasingly ossified, with even minor TCP extensions taking several years to deploy, and remarkably diverse, with applications asserting their own behaviour on how TCP performs. The legacy of middlebox support dictates that any new transport protocol will effectively be implemented over UDP or as extensions to TCP [HNR+11], such as the MPTCP. Neither of these approaches however tackles how to gain sufficient critical mass to ensure timely deployment. A clear line of future work is instead to push the transport stack up to user space, where the functionality provided can be tailored and potentially deployed alongside applications. While past work has addressed this challenge [TNML93, PWTR12], there is no mature solution for providing user-level, backward-compatible protocol stacks which can saturate existing line rates. At present however there is both a technical foundation for such software, through work in fast packet I/O mechanisms such as *netmap* [Riz12], and a practical need, as existing applications such as web caches attempt to scale beyond the performance afforded by general-purpose network stacks.

In pushing network state outwards to the end-host and exposing path diversity in the process, the *re-feedback* mechanisms explored in this thesis can be applied in exploring all of the above. The versatility of path re-feedback in particular stands as likely the single most valuable legacy of this work.

Compared to both the ToS or DS fields, it is in many ways a stronger ideological heir to the original design philosophy of the DARPA internet protocols. In deliberating on the success of the latter in [Cla88], Clark concludes by identifying shortcomings of the datagram model given requirements which had not originally been contemplated:

> *While the datagram has served very well in solving the most important goals of the Internet, it has not served so well when we attempt to address some of the goals which were further down the priority list. For example, the goals of resource management and accountability have proved difficult to achieve in the context of datagrams. As the previous section discussed, most datagrams are a part of some sequence of packets from source to destination, rather than isolated units at the application level. However, the gateway cannot directly see the existence of this sequence, because it is forced to deal with each packet in isolation. Therefore, resource management decisions or accounting must be done on each packet separately. Imposing the datagram model on the internet layer has deprived that layer of an important source of information which it could use in achieving these goals.*

In the intervening quarter of a century, progress in resource management and accountability has become predominantly driven by operators intent in coaxing existing practices from the telephone network. Rather than assessing such mechanisms in terms of their impact on the evolution of the Internet, practices such as the maintenance of flow state within the network or the prescription of classes of service are instead often justified on technical feasibility alone. For this reason, the long term ramifications of software-defined networking are particularly hard to discern given it offers a cleaner abstraction for layering and composing many new forms of network functionality – some of which may add further complexity to the datagram model. In establishing future research directions for retaining the inherent properties of the Internet while accommodating resource management in [Cla88], Clark concludes:

> *It would be necessary for the gateways to have flow state in order to remember the nature of the flows which are passing through them, but the state information would not be critical in maintaining the desired type of service associated with the flow. Instead, that type of service would be enforced by the end points, which would periodically send messages to ensure that the proper type of service was being associated with the flow. (...) I call this concept "soft state," and it may very well permit us to achieve our primary goals of survivability and flexibility, while at the same time doing a better job of dealing with the issue of resource management and accountability.*

This thesis went one step further, realising soft state as a workable technology rather than an architectural talking point and demonstrating its practicality through direct application to addressing existing requirements. However, convincing a wider community that mechanisms such as path re-feedback can greatly simplify traffic management remains unrealized and arguably the most significant outstanding challenge within this work.

Appendix A

Acronyms and Abbreviations

ABR	Available Bit Rate
ACK	acknowledgement
AFCT	Average Flow Completion Time
AIMD	Additive Increase Multiplicative decrease
AIR	Additive increase to rate
ALTO	Application Layer Traffic Optimization
API	Application Programming Interface
APNIC	Asia-Pacific Network Information Centre
AQM	Active Queue Management
ARPANET	Advanced Research Projects Agency Network
AS	Autonomous System
ASN	Autonomous System Number
ATM	Asynchronous Transfer Mode
BIC	Binary Increase Congestion Control
BGP	Border Gateway Protocol
CaTE	Content-aware Traffic Engineering
CBR	Constant Bit Rate
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CE	Congestion Experienced

CONEX Congestion Exposure

CTCP Compound TCP

cwnd congestion window

CV coefficient of variation

DCTCP Data Center TCP

DARPA Defense Advanced Research Projects Agency

DCCP Datagram Congestion Control Protocol

DEC Digital Equipment Corporation

DNS Domain Name System

DoS Denial-of-service

DPI Deep Packet Inspection

DS Differentiated Services

EC Efficiency Controller

ECN Explicit Congestion Notification

ECT ECN Capable Transport

ECMP Equal Cost Multipath

EFCI Explicit Forward Congestion Indication

FAST Fast AQM Scalable TCP

FC Fairness Controller

FEC Forwarding Equivalence Class

FFT Fast Fourier Transform

FLARE Flowlet Aware Routing Engine

FNE Feedback Not Established

FRR Fast Re-Route

GMT Greenwich Meridian Time

HTTP Hyper Text Transfer Protocol

ICMP Internet Control Message Protocol

IIJ Internet Initiative Japan

INF Interior Forwarding

IS-IS Intermediate System to Intermediate System

IETF Internet Engineering Task Force

IGP Interior Gateway Protocol

ILNP Identifier Locator Naming Protocol

IMP Interface Message Processor

I/O Input/Output

IP Internet Protocol

IPv4 IP Version 4

ISP Internet Service Provider

IXP Internet Exchange Point

JST Japan Standard Time

LAN Local Area Network

LECT Loss Exposure Capable Transport

LEx Loss Experienced

LEX Loss Exposure

LFA Loop-Free Alternate

LISP Locator/Identifier Separation Protocol

LSP Label Switched Path

LSR Label Switching Router

LSRR Loose Source and Record Route

MATE Multipath Adaptive TE

MAWI Measurement and Analysis of the WIDE Internet

MCR Minimum Cell Rate

MED Multi-Exit Discriminator

MPLS Multi Protocol Label Switching

MPTCP Multipath TCP

MRC Multiple Routing Configurations

MSS Maximum Segment Size

NAT Network Address Translator

NCP Network Control Program

NPL National Physical Laboratory, UK

NTT Nippon Telegraph & Telephone Corporation

OCH One-click Hosting

OSPF Open Shortest Path First

OS operating system

P2P Peer-to-peer

P4P Provider portal for applications

PCR Peak Cell Rate

PEP Performance Enhancing Proxy

PREF Path Re-Feedback

PREFLEX Path Re-Feedback and Loss Exposure

QoS Quality of Service

RAND Research And Development Corporation

RECT Re-ECN Capable Transport

RIB Routing Information Base

RCP Rate Control Protocol

RDF Rate decrease factor

RED Random Early Detection

RIR Routing Information Registrar

RM Resource Management

RON Resilient Overlay Routing

RTO retransmission timeout

RTT round trip time

SACK Selective acknowledgement

SCTP Stream control transport protocol

SDN Software-defined networking

ssthresh slow-start threshold

SOSR Scalable One-hop Source Routing

TCAM Ternary Content-addressable Memory

TE Traffic Engineering

TeXCP TE with XCP

TM Traffic Matrix

ToS Type of Service

TCP Transmission Control Protocol

UDP User Datagram Protocol

VBR Variable Bit Rate

VCP Variable-structure congestion Control Protocol

VLAN Virtual LAN

VoIP Voice-over-IP

WIDE Widely Integrated Distributed Environment

WFQ Weighted Fair Queuing

XCP eXplicit Congestion Protocol

Bibliography

[ABH09] R Atkinson, S Bhatti, and S Hailes. ILNP: mobility, multi-homing, localised addressing and security through naming. *Telecommunication Systems*, 42(3):273–291, 2009.

[ABKM02] D Andersen, H Balakrishnan, F Kaashoek, and R Morris. Resilient overlay networks. *ACM SIGCOMM Computer Communication Review*, 32(1):66–66, 2002.

[ACF+12] B Ager, N Chatzis, A Feldmann, N Sarrar, and S Uhlig. Anatomy of a Large European IXP. *SIGCOMM '12: Proceedings of the 2012 conference on Applications, technologies, architectures, and protocols for computer communications*, 2012.

[AGM+10] M Alizadeh, A Greenberg, D.A Maltz, J Padhye, P Patel, B Prabhakar, S Sengupta, M Sridharan, C Faster, and D Maltz. DCTCP: Efficient packet transport for the commoditized data center. *SIGCOMM '10: Proceedings of the 2010 conference on Applications, technologies, architectures, and protocols for computer communications*, 2010.

[AMD09] Demetris Antoniades, Evangelos P. Markatos, and Constantine Dovrolis. One-click hosting services: a file-sharing hideout. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 223–234, New York, NY, USA, 2009. ACM.

[AMSU11] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. Web content cartography. In *Proceedings of the 11th ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 585–600, New York, NY, USA, 2011. ACM.

[AN11] S Alcock and R Nelson. Application flow control in youtube video streams. *ACM SIGCOMM Computer Communication Review*, 41(2):24–30, 2011.

[app] AppEx IPEQ (IP End-to-End QoS). `http://www.appexnetworks.com/white-papers/IPEQ.pdf`.

[Atl06] A Atlas. U-turn Alternates for IP/LDP Fast-Reroute. *IETF Internet draft*, draft-atlas-ip-local-protect-uturn-03, 2006.

[Bak95] F Baker. RFC1812: Requirements for IP version 4 routers. *IETF Request for Comments*, 1995.

[Bar64a] P Baran. ON DISTRIBUTED COMMUNICATIONS: IV. PRIORITY, PRECEDENCE, AND OVERLOAD. *rand.org*, Jan 1964.

[Bar64b] P Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, 1964.

[BBB11] S Bauer, R Beverly, and A Berger. Measuring the state of ECN readiness in servers, clients, and routers. *Proceedings of the 11th ACM SIGCOMM conference on Internet measurement*, pages 171–180, 2011.

[BBC+98] S Blake, D Black, M Carlson, E Davies, Z Wang, and W Weiss. RFC 2475: An architecture for differentiated services. 1998.

[BCL09] S Bauer, D Clark, and W Lehr. The evolution of internet congestion. *Proceedings of the 37th Research Conference on Communication, Information, and Internet Policy*, 2009.

[BF95] F Bonomi and K.W Fendick. The rate-based flow control framework for the available bit rate ATM service. *IEEE Network*, 9(2):25–39, 1995.

[BFPS07] S Bryant, C Filsfils, S Previdi, and M Shand. IP Fast Reroute using tunnels. *IETF Internet draft*, draft-bryant-ipfrr-tunnels-03, 2007.

[BJCG+05] B Briscoe, A Jacquet, C Di Cairano-Gilfedder, A Salvatori, A Soppera, and M Koyabe. Policing congestion response in an internetwork using re-feedback. *ACM SIGCOMM Computer Communication Review*, 35(4):277–288, 2005.

[BJMS08] B Briscoe, A Jacquet, T Moncaster, and A Smith. Re-ECN: Adding accountability for causing congestion to TCP/IP. *IETF Internet draft*, draft-briscoe-tsvwg-re-ecn-tcp-05.txt, 2008.

[BKL+09] C Bastian, T Klieber, J Livingood, J Mills, and R Woundy. RFC6057: Comcast's Protocol-Agnostic Congestion Management System. *IETF Request for Comments*, 2009.

[BLNS81] A.D Birrell, R Levin, R.M Needham, and M.D Schroeder. Grapevine: An exercise in distributed computing. *ACM SIGOPS Operating Systems Review*, 15(5):178–179, 1981.

[BOP94] L.S Brakmo, S.W O'malley, and L.L Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *ACM SIGCOMM Computer Communication Review*, 24(4):24–35, 1994.

[BPS99] JCR Bennett, C Partridge, and N Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, 1999.

[Bra89] Robert Braden. RFC-1122: Requirements for internet hosts. *Request for Comments*, pages 356–363, 1989.

[Bri95] T Brisco. RFC1794: DNS support for load balancing. *IETF Request for Comments*, 1995.

[Bri07] B Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, 2007.

[Bus05] R Bush. Into the future with the internet vendor task force a very curmudgeonly view or testing spaghetti: a wall's point of view. *ACM SIGCOMM Computer Communication Review*, 35(5):67–68, 2005.

[CAI] Cooperative Association for Internet Data Analysis (CAIDA). `http://www.caida.org/`.

[CAL+13] Richard G. Clegg, Joao Taveira Araujo, Raul Landa, Eleni Mykoniati, David Griffin, and Miguel Rio. On the relationship between fundamental measurements in TCP flows. In *Proceedings of IEEE ICC*, 2013.

[CB08] D.R Choffnes and F.E Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[Cer74] V Cerf. RFC635: Assessment of ARPANET protocols. *IETF Request for Comments*, 1974.

[CFEK06] K Cho, K Fukuda, H Esaki, and A Kato. The impact and implications of the growth in residential user-to-user traffic. *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, page 218, 2006.

[CFEK08] K Cho, K Fukuda, H Esaki, and A Kato. Observing slow crustal movement in residential user traffic. *Proceedings of the 4th COnference on emerging Networking EXperiments and Technologies (CoNEXT)*, page 12, 2008.

[CHM+03] J Crowcroft, S Hand, R Mortier, T Roscoe, and A Warfield. QoS's downfall: at the bottom, or not at all! *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?*, pages 109–114, 2003.

[CI05] V.G Cerf and R.E Icahn. A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, 35(2):71–82, 2005.

[CL05] R.K.C Chang and M Lo. Inbound traffic engineering for multihomed ASs using as path prepending. *IEEE Network*, 19(2):18–25, 2005.

[Cla88] D Clark. The design philosophy of the DARPA Internet protocols. *ACM SIGCOMM Computer Communication Review*, 18(4):106–114, 1988.

[CLRS10] P Chhabra, N Laoutaris, P Rodriguez, and R Sundaram. Home is where the (fast) internet is: flat-rate compatible incentives for reducing peak load. *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, pages 13–18, 2010.

[CMK00] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the WIDE project. In *Proceedings of the USENIX Annual Technical Conference*, 2000.

[CWSB05] David Clark, John Wroclawski, Karen Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Transactions on Networking*, 13(3), Jun 2005.

[Dav72] D Davies. The control of congestion in packet-switching networks. *IEEE Transactions on Communications*, 20(3):546–550, 1972.

[Day10] John Day. *Patterns in Network Architecture: A Return to Fundamentals*. Jan 2010.

[DD11] A Dhamdhere and C Dovrolis. Twelve Years in the Evolution of the Internet Ecosystem. *IEEE/ACM Transactions on Networking*, 19(5):1420 – 1433, 2011.

[DKZSM05] N Dukkipati, M Kobayashi, R Zhang-Shen, and N McKeown. Processor sharing flows in the internet. *Quality of Service–IWQoS 2005*, pages 271–285, 2005.

[DMG+10] M Dischinger, M Marcon, S Guha, KP Gummadi, R Mahajan, and S Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. *Proceedings of 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[DRCC10] N Dukkipati, T Refice, Y Cheng, and J Chu. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM Computer Communication Review*, Jan 2010.

[EJLW02] A Elwalid, C Jin, S Low, and I Widjaja. MATE: multipath adaptive traffic engineering. *Computer Networks*, 40(6):695–709, 2002.

[FA08] S Floyd and M Allman. RFC5290: Comments on the Usefulness of Simple Best-Effort Traffic. *IETF Request for Comments*, 2008.

[FBAF10] R Fontugne, P Borgnat, P Abry, and K Fukuda. MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. *Proceedings of the 6th COnference on emerging Networking EXperiments and Technologies (CoNEXT)*, page 8, 2010.

[FFEB05] P Francois, C Filsfils, J Evans, and O Bonaventure. Achieving sub-second IGP convergence in large IP networks. *ACM SIGCOMM Computer Communication Review*, 35(3):35–44, 2005.

[FGL+00] A Feldmann, A Greenberg, C Lund, N Reingold, and J Rexford. Netscope: traffic engineering for IP networks. *IEEE Network*, 14(2):11–19, 2000.

[FI08] B Ford and J Iyengar. Breaking up the transport logjam. *7th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2008.

[FJ93] S Floyd and V Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[Flo94] S Floyd. TCP and explicit congestion notification. *ACM SIGCOMM Computer Communication Review*, 24(5):8–23, 1994.

[FRH+11] A Ford, C Raiciu, M Handley, S Barré, and J Iyengar. RFC6182: Architectural guidelines for multipath TCP development. *IETF Request for Comments*, 2011.

[FT00] B Fortz and M Thorup. Internet traffic engineering by optimizing OSPF weights. *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM)*, 2:519–528 vol. 2, 2000.

[FT02] B Fortz and M Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

[GCX+05] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, IMC '05, pages 4–4, Berkeley, CA, USA, 2005. USENIX Association.

[GGSS09] P Godfrey, I Ganichev, S Shenker, and I Stoica. Pathlet routing. *ACM SIGCOMM Computer Communication Review*, 39(4):111–122, 2009.

[GMG+04] KP Gummadi, HV Madhyastha, SD Gribble, HM Levy, and D Wetherall. Improving the reliability of internet paths with one-hop source routing. *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, page 13, 2004.

[GQX+04] David Goldenberg, Lili Qiuy, Haiyong Xie, Yang Yang, and Yin Zhang. Optimizing cost and performance for multihoming. *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug 2004.

[GRF03] M Goyal, KK Ramakrishnan, and W Feng. Achieving faster failure detection in OSPF networks. *Proceedings of the 2003 IEEE International Conference on Communications (ICC)*, 1:296–300 vol. 1, 2003.

[Hat73] W Hathaway. RFC512: More on lost message detection. *IETF Request for Comments*, Jan 1973.

[HCR06] J He, M Chiang, and J Rexford. Can Congestion Control and Traffic Engineering Be at Odds? *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–6, 2006.

[HFP+05] M.P Howarth, P Flegkas, G Pavlou, N Wang, P Trimintzios, D Griffin, J Griem, M Boucadair, P Morand, and A Asgari. Provisioning for interdomain quality of service: the MESCAL approach. *IEEE Communications Magazine*, 43(6):129–137, 2005.

[HFU+10] H Haddadi, D Fay, S Uhlig, A Moore, R Mortier, and A Jamakovic. Mixing Biases: Structural Changes in the AS Topology Evolution. *Traffic Monitoring and Analysis*, pages 32–45, 2010.

[HLM+04] N Hu, LE Li, ZM Mao, P Steenkiste, and J Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 41–54, 2004.

[HNR+11] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 181–194, New York, NY, USA, 2011. ACM.

[HRX08] S Ha, I Rhee, and L Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[HS02] H.Y Hsieh and R Sivakumar. pTCP: An end-to-end transport layer protocol for striped connections. *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*, pages 24–33, 2002.

[HSM12] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *1st ACM SIGCOMM Workshop on Hot Topics in Software-defined Networks (HotSDN)*, 2012.

[Hui95] C Huitema. Multi-homed TCP. *IETF Internet draft*, draft-huitema-multi-homed-01, 1995.

[IAS06] J.R Iyengar, P.D Amer, and R Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, 2006.

[Jac88] V Jacobson. Congestion avoidance and control. *SIGCOMM '88: Proceedings of the 1988 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug 1988.

[Jai96] R Jain. Congestion control and traffic management in ATM networks: Recent advances and a survey. *Computer networks and ISDN systems*, 28(13):1723–1738, 1996.

[JBB92] Van Jacobson, Robert Braden, and David Borman. TCP extensions for high performance. RFC1323, 1992.

[JBM08] A Jacquet, B Briscoe, and T Moncaster. Policing freedom to use the internet resource pool. *Proceedings of the 4th COnference on emerging Networking EXperiments and Technologies (CoNEXT)*, page 71, 2008.

[JID+04] S Jaiswal, G Iannaccone, C Diot, J Kurose, and D Towsley. Inferring TCP connection characteristics through passive measurements. *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM)*, 3:1582–1592 vol. 3, 2004.

[JID+07] S Jaiswal, G Iannaccone, C Diot, J Kurose, and D Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Transactions on Networking*, 15(1):54–66, 2007.

[JWHC11] C Joe-Wong, S Ha, and M Chiang. Time-dependent broadband pricing: Feasibility and benefits. *Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS)*, pages 288–298, 2011.

[KD11] Partha Kanuparthy and Constantine Dovrolis. Shaperprobe: end-to-end detection of isp traffic shaping using active methods. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 473–482, New York, NY, USA, 2011. ACM.

[Kel03] T Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[KF07] D Katabi and Aaron Falk. Specification for the Explicit Control Protocol (XCP). *IETF Internet draft*, draft-falk-xcp-spec-03. txt, 2007.

[KHR02] D Katabi, M Handley, and C Rohrs. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM Computer Communication Review*, 32(4):89–102, 2002.

[KKDC05] S Kandula, D Katabi, B Davie, and A Charny. Walking the tightrope: Responsive yet stable traffic engineering. *ACM SIGCOMM Computer Communication Review*, 35(4):264, 2005.

[KL00] M Kodialam and TV Lakshman. Minimum interference routing with applications to MPLS traffic engineering. *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM)*, 2:884–893 vol. 2, 2000.

[Kle75] Leonard Kleinrock. Queueing Systems. I: Theory, 1975.

[KMT98] FP Kelly, AK Maulloo, and DKH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[KMT07] P Key, L Massoulie, and P.D Towsley. Path Selection and Multipath Congestion Control. *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 143–151, 2007.

[Kuz05] A Kuzmanovic. The power of explicit congestion notification. *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, page 72, 2005.

[KV05] F Kelly and T Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):12, 2005.

[LF05] R Lance and I Frommer. Round-trip time inference via passive monitoring. *ACM SIGMETRICS Performance Evaluation Review*, 33(3):32–38, 2005.

[LFFM12] D Lewis, V Fuller, D Farinacci, and D Meyer. Locator/ID Separation Protocol (LISP). *IETF Internet Draft*, draft-ietf-lisp-23, Jan 2012.

[LH06] K Lan and J Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46–62, 2006.

[LIJM+10a] C Labovitz, S Iekel-Johnson, D McPherson, J Oberheide, and F Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 40(4):75–86, 2010.

[LIJM+10b] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 41(4):–, August 2010.

[LJC08] P Laskowski, B Johnson, and J Chuang. User-directed routing: from theory, towards practice. *Proceedings of the 3rd international workshop on Economics of networked systems*, pages 1–6, 2008.

[LMJ98] C Labovitz, G.R Malan, and F Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, 1998.

[Lot92] M Lottor. RFC1296: Internet Growth (1981-1991). *IETF Request for Comments*, 1992.

[LR08] N Laoutaris and P Rodriguez. Good Things Come to Those Who (Can) Wait or how to handle Delay Tolerant traffic and make peace on the Internet. *7th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2008.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2), March 2008.

[MABW09] T Moncaster, J Araujo, S Blake, and R Woundy. The Need for Congestion Exposure in the Internet. *IETF Internet draft*, draft-moncaster-congestion-exposure-problem-01, 2009.

[Mat09] M Mathis. Relentless congestion control. *Prooceedings of PFLDNet*, 2009.

[max12] MaxMind GeoLite City. http://www.maxmind.com/app/geolitecity, 2012.

[MEFV08] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, Aug 2008.

[MMC00] H. S. Martin, A. McGregor, and J. G. Cleary. Analysis of internet delay times. In *Proceedings of the Passive and Active Measurements Workshop (PAM)*, 2000.

[MMV95] J.K MacKie-Mason and H.R Varian. Pricing congestible network resources. *IEEE Journal on Selected Areas in Communications*, 13(7):1141–1149, 1995.

[Moc87] P Mockapetris. RFC 1035: Domain Names-Implementation and Specification. *IETF Request for Comments*, 1987.

[MZPP08] R Mahajan, M Zhang, L Poole, and V Pai. Uncovering performance differences among backbone ISPs with Netdiff. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 205–218, 2008.

[Nag84] J Nagle. RFC896: Congestion control in IP/TCP internetworks. *IETF Request for Comments*, 1984.

[ns3] Network Simulator 3. `http://www.nsnam.org`.

[Odl04] A Odlyzko. Pricing and architecture of the Internet: Historical perspectives from telecommunications and transportation. *Proceedings of the 32th Research Conference on Communication, Information, and Internet Policy*, 2004.

[OZP+06] R Oliveira, B Zhang, D Pei, R Izhak-Ratzin, and L Zhang. Quantifying path exploration in the internet. *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 269–282, 2006.

[OZPZ09] R Oliveira, B Zhang, D Pei, and L Zhang. Quantifying path exploration in the internet. *IEEE/ACM Transactions on Networking*, 17(2):445–458, 2009.

[PAS99] V Paxson, M Allman, and W Stevens. RFC2581: TCP congestion control. *IETF Request for Comments*, 1999.

[PC09] J Peterson and A Cooper. Report from the IETF Workshop on Peer-to-Peer (P2P) Infrastructure, May 28, 2008. *Center for Democracy & Technology*, 2009.

[PFS+12] Ingmar Poese, Benjamin Frank, Georgios Smaragdakis, Steve Uhlig, Anja Feldmann, and Bruce Maggs. Enabling content-aware traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 42(5):21–28, September 2012.

[Pos73] J Postel. RFC516: Lost message detection. *IETF Request for Comments*, Jan 1973.

[Pos80] J Postel. DoD standard transmission control protocol. 1980.

[Pos81] J Postel. Internet control message protocol. 1981.

[Pou73] L Pouzin. Presentation and major design aspects of the CYCLADES computer network. *NATO Advanced Study Institute on Computer Communication Networks*, 1973.

[pox] POX OpenFlow Controller. `http://www.noxrepo.org/pox`.

[PPK+09] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Open vSwitch: Extending networking into the virtualization layer. In *8th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2009.

[PSA05] P Pan, G Swallow, and A Atlas. RFC4090: Fast reroute extensions to RSVP-TE for LSP tunnels. *RFC4090, May*, 2005.

[PWTR12] B. Penoff, A. Wagner, M. Tuxen, and I. Rungeler. Portable and Performant Userspace SCTP Stack. In *Proceedings of the 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2012.

[QGM+09] F Qian, A Gerber, Z.M Mao, S Sen, O Spatscheck, and W Willinger. TCP revisited: a fresh look at TCP in the wild. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 76–89, 2009.

[QPS+03] B Quoitin, C Pelsser, L Swinnen, O Bonaventure, and S Uhlig. Interdomain traffic engineering with BGP. *IEEE Communications Magazine*, 41(5):122– 128, 2003.

[QTUB04] B Quoitin, S Tandel, S Uhlig, and O Bonaventure. Interdomain traffic engineering with redistribution communities. *Computer Communications*, 27(4):355–363, 2004.

[RCC+11] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. Tcp fast open. In *Proceedings of the 2011 ACM CoNEXT Conference*, CoNEXT '11, pages 21:1–21:12, New York, NY, USA, 2011. ACM.

[RCG+10] Fernando M. V. Ramos, Jon Crowcroft, Richard J. Gibbens, Pablo Rodriguez, and Ian H. White. Channel smurfing: Minimising channel switching delay in iptv distribution networks. In *ICME*, pages 1327–1332. IEEE, 2010.

[RFB01] K Ramakrishnan, S Floyd, and D Black. RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP. *IETF Request for Comments*, 2001.

[Riz12] L. Rizzo. Revisiting Network I/O APIs: The netmap Framework. *ACM Queue*, 10(1):30–39, Jan. 2012.

[RJ90] KK Ramakrishnan and R Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems (TOCS)*, 8(2):158–181, 1990.

[RKS07] S Rewaskar, J Kaur, and FD Smith. A performance study of loss detection/recovery in real-world TCP implementations. *Proceedings of the 15th IEEE International Conference on Network Protocols (ICNP)*, pages 256–265, 2007.

[RLL+11] A Rao, A Legout, Y Lim, D Towsley, C Barakat, and W Dabbous. Network characteristics of video streaming traffic. *Proceedings of the 7th COnference on emerging Networking EXperiments and Technologies (CoNEXT)*, page 25, 2011.

[RNS+12] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *1st ACM SIGCOMM Workshop on Hot Topics in Software-defined Networks (HotSDN)*, 2012.

[rou] Route Views Project Page. `http://www.routeviews.org/`.

[RVC01] E Rosen, A Viswanathan, and R Callon. RFC3031: Multiprotocol Label Switching Architecture. *IETF Request for Comments*, 2001.

[SB10] M Shand and S Bryant. RFC5714: IP Fast Reroute Framework. *IETF Request for Comments*, 2010.

[SCBRSP12] Josep Sanjuàs-Cuxart, Pere Barlet-Ros, and Josep Solé-Pareta. Measurement based analysis of one-click file hosting services. *Journal of Network and Systems Management*, 20(2):276–301, 2012.

[SKK04] S Sinha, S Kandula, and D Katabi. Harnessing TCP's burstiness with flowlet switching. *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2004.

[SPB11] M Shand, S Previdi, and S Bryant. IP fast reroute using not-via addresses. *IETF Internet Draft*, draft-ietf-rtgwg-ipfrr-notvia-addresses-09, 2011.

[SRC84] J.H Saltzer, D.P Reed, and D.D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[SSB+04] S Shakkottai, R Srikant, N Brownlee, A Broido, and K C Claffy. The RTT distribution of TCP flows in the Internet and its impact on TCP-based flow control. *Tech Report Cooperative Association for Internet Data Analysis (CAIDA)*, 2004.

[Ste97] W.R Stevens. RFC2001: TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. *IETF Request for Comments*, 1997.

[Ste07] R. Stewart. RFC4960: Stream Control Transmission Protocol, September 2007. Updated by RFCs 6096, 6335.

[Sun77] CA Sunshine. Source routing in computer networks. *ACM SIGCOMM Computer Communication Review*, 7(1):29–33, 1977.

[TAC+08] R Torvi, A Atlas, G Choudhury, A Zinin, and B Imhoff. RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates. *IETF Request for Comments*, Jan 2008.

[TH00] D Thaler and C Hopps. RFC2991: Multipath Issues in Unicast and Multicast Next-Hop Selection. *IETF Request for Comments*, 2000.

[Tha10] Dave Thaler. Evolution of the IP Model. *IETF Internet draft*, draft-iab-ip-model-evolution-02, 2010.

[TMSV03] R Teixeira, K Marzullo, S Savage, and GM Voelker. In search of path diversity in ISP networks. *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, page 318, 2003.

[TNML93] C. Thekkath, T. Nguyen, E. Moy, and E. Lazowska. Implementing network protocols at user level. *IEEE/ACM Transactions on Networking*, 1(5):554–565, 1993.

[TS06] Kun Tan and Jingmin Song. A compound tcp approach for high-speed and long distance networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[UB04] S Uhlig and O Bonaventure. Designing BGP-based outbound traffic engineering techniques for stub ASes. *ACM SIGCOMM Computer Communication Review*, 34(5):106, 2004.

[vis] Description of the Receive Window Auto-Tuning feature for HTTP traffic on Windows Vista-based computers. `http://support.microsoft.com/kb/947239`.

[VLL05] B Veal, K Li, and D Lowenthal. New methods for passive estimation of TCP round-trip times. *Proceedings of Passive and Active Measurment Workshop (PAM)*, pages 121–134, 2005.

[WHB08] D Wischik, M Handley, and MB Braun. The resource pooling principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

[WHPH08] Ning Wang, Kin Ho, G Pavlou, and M Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.

[WJLH06] D.X Wei, C Jin, S.H Low, and S Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006.

[WRGH11] D Wischik, C Raiciu, A Greenhalgh, and M Handley. Design, implementation and evaluation of congestion control for multipath TCP. *Proceedings of 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[WW99] Yufei Wang and Zheng Wang. Explicit routing algorithms for Internet traffic engineering. *Proceedings of the 8th International Conference on Computer Communications and Networks (ICCCN)*, pages 582–588, 1999.

[WWZ01] Y Wang, Z Wang, and L Zhang. Internet traffic engineering without full mesh overlaying. *Proceedings of the 20th IEEE International Conference on Computer Communications (INFOCOM)*, 1:565–571 vol. 1, 2001.

[XHBN00] X Xiao, A Hannan, B Bailey, and L.M Ni. Traffic Engineering with MPLS in the Internet. *IEEE Network*, 14(2):28–33, 2000.

[XHR04] L Xu, K Harfoush, and I Rhee. Binary increase congestion control (BIC) for fast long-distance networks. *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM)*, 4:2514–2524 vol. 4, 2004.

[XR06] Wen Xu and Jennifer Rexford. Miro: multi-path interdomain routing. *ACM SIGCOMM Computer Communication Review*, 36(4):171–182, August 2006.

[XSSK08] Yong Xia, L Subramanian, I Stoica, and S Kalyanaraman. One More Bit is Enough. *IEEE/ACM Transactions on Networking*, 16(6):1281–1294, 2008.

[XYK+08] H Xie, Y.R Yang, A Krishnamurthy, Y.G Liu, and A Silberschatz. P4p: provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[Yan03] X Yang. NIRA: A new Internet routing architecture. *ACM SIGCOMM Computer Communication Review*, 33(4):312, 2003.

[YW06] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug 2006.

[ZBPS02] Y Zhang, L Breslau, V Paxson, and S Shenker. On the characteristics and origins of internet flow rates. *ACM SIGCOMM Computer Communication Review*, 32(4):322, 2002.