On Flow Concurrency in the Internet and its Implications for Capacity Sharing

Brian Trammell
Communication Systems Group
ETH Zurich, Switzerland
trammell@tik.ee.ethz.ch

Dominik Schatzmann
Communication Systems Group
ETH Zurich, Switzerland
schatzmann@tik.ee.ethz.ch

ABSTRACT

Flow concurrency is the measure of the number of active flows at a given point in time at a given point in the network, and can be used as a complement to traffic volume to understand the dynamics of a measured network. It is of particular interest given the spread of devices through the network which keep per-flow state. In this work, we first present a simple methodology for measuring flow concurrency using network flow data, then apply this methodology to a long-term data archive captured at the border of a national-scale research network to measure flow concurrency in selected example network configurations (e.g., at a content consumer network, on a content provider network, at an interconnect point).

Flow concurrency is interesting in the context of capacity sharing efforts in two ways. First and most obviously, devices which verify and enforce policy compliance for capacity sharing must keep per-flow state on either end of a flow; flow state requirements therefore dictate where such devices may be placed in the network, and the trust properties of the algorithms they run. Second, as the deployment of flow-state-keeping devices in the network increases, flow state itself becomes a "congestible" resource just as queue space is: future work in capacity sharing may consider addressing this.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

Keywords

Internet measurement, flow concurrency, fairness

1. INTRODUCTION

Many devices deployed within today's Internet must keep per-flow state, and their performance is therefore a function not just of the observed or handled packet rate, but of the number of concurrent flows as well. This class of devices includes network address translators, many packet-inspecting middleboxes such as stateful firewalls, and monitoring and measurement devices such as flow meters and intrusion detection devices. The use of such devices is growing in the Internet.

This work provides a simple methodology for measuring flow concurrency in section 2, which is related to per-flow state requirements for a given application on a given link within a network. We then aim to give general guidance for flow state requirements for given network types in section 3, by applying this methodology to a NetFlow data set collected at the border of a national-scale research network to derive trends in flow concurrency for networks of various types.

"Rules of thumb" derived from community experience can be used in designing and provisioning these devices to handle the traffic on the networks on which they are deployed; these generally err on the side of massive over-provisioning. However, this work shows that heterogeneity of flow concurrency both in space and time make it difficult to simply predict state requirements at a given measurement point. In addition, changes in application usage patterns over time may break implicit assumptions made by these estimations.

This work follows a long line of Internet traffic studies showing the growth of traffic and characteristics of traffic on different networks [11, 3]. Flow concurrency as a metric in itself is not directly treated by these works. However, there is a wide body of work treating flow concurrency implicitly, as it based on analysis of the concurrent flow set at a given observation point, especially for network security and troubleshooting applications. For example, Tellenbach et al. [14] identify anomalies such as DDoS attacks through analysis of the set of concurrent flows. FACT [13] is an online troubleshooting application that helps network operators to track connectivity problems occurring in remote autonomous systems, networks, and hosts by analyzing the set of concurrent flows.

As for the implications of these findings for capacity sharing efforts, this work was initially inspired by ongoing work in the IETF conex working group [5]. The authors realized the policy verification and enforcement functions required by the protocol, described abstractly in [12], have per-flow state requirements. This follows earlier work in the area, e.g. [4], which implicitly evaluates flow state requirements, though the performance evaluation of the audit function is primarily focused on its correctness in terms of fairness benefit provided.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CSWS'12, December 10, 2012, Nice, France.
Copyright 2012 ACM 978-1-4503-1780-1/12/12 ...$15.00.

The algorithms proposed to date only keep state close to the edge, thereby avoiding links with very high flow concurrency; this work provides measurements to support this choice. We discuss this further in section 4.1, which comes to the admittedly unsurprising conclusion that at current levels of flow concurrency, policy verification and enforcement at the edge is scalable, but algorithms which must be placed at large network exchange points should continue to be avoided. This implies that in multi-provider networks, audit algorithms must function without requiring trust between components.

The problem of flow state proliferation as in section 4.2 has been addressed from several different angles in recent literature. The realization that flow-state-related resources at middleboxes should be subject to fairness in addition to bandwidth has been treated in [9]; here, the authors present a scheduling algorithm for middleboxes which provides superior throughput by accounting for state and CPU separately from bandwidth, and enforcing fair allocation thereof. Additionally, Honda et al. [10] evaluated such middleboxes for their impact on various features of TCP, and [7] focuses on end-user visible application performance and functionality impact. While neither of these address the flow-state problem directly, the methodologies there could be adapted and augmented to begin to estimate the amount of state available along a path.

2. METHODOLOGY

Simulations or direct measurements of flow-state-keeping devices can measure flow concurrency trivially: the size of the flow table is a necessary variable for the operation of such devices, and can be directly observed. However, when estimating flow concurrency from traffic traces at the packet or flow level, some analysis is necessary.

The methodology we apply in this work is straightforward. We start with unsampled flow data taken from six routers at the border of SWITCH[1], the Swiss national research and education network (NREN). This network originates about 2.4M IPv4 addresses (the rough equivalent of a /11). Typical daily traffic volume is between 50 and 100 TB. This flow data includes the standard "5-tuple" (source and destination address and port, plus protocol), as well as start and end timestamps with an effective resolution of seconds[2]; there are additional fields present concerning flow treatment (interface, BGP information, next-hop address) which we do not consider in this work.

We maintain a rolling window of n bins of a set interval t, such that $n \cdot t > TO_{active} + D_{max}$, where TO_{active} is the *active timeout* of the flow collection infrastructure (i.e., the time after which any flow will be expired from the flow meter's cache, or alternately, the maximum duration of any exported flow) and D_{max} is the maximum observed export delay. Therefore, we are guaranteed that any observed flow can be assigned to all bins it covers. This interval t represents the maximum amount of time flow state will be unnecessarily kept for a closed flow.

For each flow, we add its 5-tuple flow key to the set of known flows for the bin containing its start time, the bin containing its end time, and all bins in between. When a bin expires out of the rolling window, we then simply count the flow keys in the set. This produces a non-conservative flow count as defined in [15].

[1]www.switch.ch

[2]The flow metering technology used has an inherent limitation on timing resolution of one to two seconds, as detailed in [16]; the corrections described in that paper are *not* applied to the source data in this work.

We note that while we chose 5-tuple flow keys, the presented methodology can be trivially generalized to determine flow concurrency for any granularity of flow key, e.g. by prefix, to estimate flow state requirements for applications at higher degrees of aggregation.

We also count unique active hosts per bin inside and outside the border of the observed network, in order to normalize state requirements per active host; a host is considered active in a bin if is participates in at least one flow.

Our study is limited to TCP flows, since TCP flows are those with which capacity sharing approaches are generally concerned. In general, we choose a t of 5 seconds, as this is close to the maximum resolution of the timing information we have available, and represents a rather aggressive timeout strategy to be applied by flow-state-keeping middleboxes. This allows us to report best-case numbers, as we discuss further in section 3.1. For each given data set, we report median, 95^{th} percentile, and peak numbers – though in many cases concurrency varies widely based on diurnal seasonality, as the state requirements are generally driven by human activity, flow-state-keeping devices must be provisioned to handle peak load.

In interpreting results taken from this network, it is important to note that it is an access and interconnection network for research institutes and universities without a significant residential population, and is therefore somewhat biased toward weekday, working-hour traffic. However, we focused on several subsets of the traffic collected, in order to attempt to derive general number for flow concurrency on the wider Internet from this traffic set:

- All traffic crossing the border of the network at large; we use this as a proxy for a large interconnect point.
- Traffic crossing the border for one university using the network for Internet access, representing 5 /18s, about 80,000 addresses.
- Traffic crossing the border for a set of about 13,000 addresses on predominantly client networks at the university, and for a set of about 13,000 addresses on predominantly server networks at the university.
- Traffic crossing the border from three external ASNs representing large content provider networks.

3. FINDINGS

Our key findings are as follows:

- Flow concurrency per link / per network depends greatly on the type of network observed. Content provider networks see much greater flow concurrency than access networks, for example.
- Flow concurrency per *active host* is relatively stable given the type of activity each host is engaged in: clients generally have a concurrency of at least 4 and median concurrency of about 6 flows, while server flow concurrency is generally higher, and dependent on server popularity.

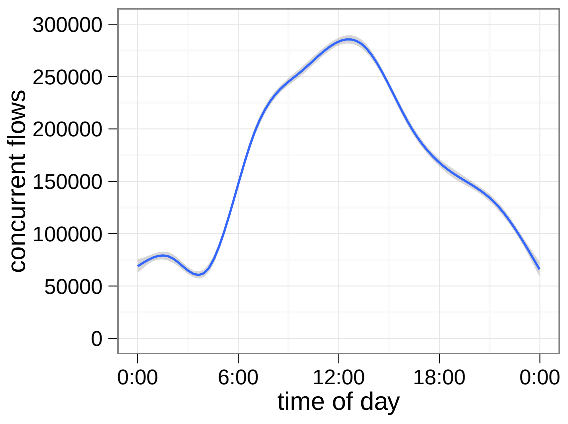

Figure 1: Diurnal rhythm in flow concurrency: smoothed, uniformly sampled, $t = 5s$

- Measured flow concurrency, and therefore state requirements for flow-state-keeping devices, increases nearly linearly with the timeout interval t chosen; this indicates that such devices should use timeouts that are as aggressive as possible.
- The trend in flow concurrency increases year over year, reflecting the use of increased network capacity.

First we examine TCP flow concurrency with $t = 5s$ for the entire measured network, which originates about 2.4 million IPv4 addresses. Over several sampled days in 2011, median flow concurrency was 148 kflows, with 95^{th} percentile 322 kflows, and peak 436 kflows. In the same bins, median utilization of advertised address space was 1.27%, 95^{th}-%ile 2.54%, and peak 10.9%[3]. On longer timescales (5 minutes), we see that about a quarter of the advertised address space is used at any given time.

This activity is strongly diurnal, as shown in 1. From 08:00 to 20:00 local time, median / 95^{th}-%ile flow concurrency is 253,000 / 343 kflows, while overnight, it falls to 91.4 / 166 kflows. However, night-time transient peak concurrency is comparable to that during the day.

Drilling down to a subset of the traffic, we examine a single university, consisting of 5 /18 networks. Here we see median flow concurrency of 3.20 kflows, 95^{th}-%ile 4.30 kflows, and peak 22.900 kflows.

This initial analysis would indicate a rule of thumb to dimension flow-state-keeping devices to handle on the order of 10,000 - 20,000 concurrent flows per /16 of address space behind the device, assuming comparable utilization, traffic mix, and timeouts to those in use on the measured network. However, as utilization and traffic mix vary widely, we decided to focus on both client and server traffic separately, to generate numbers on flow concurrency per *active* host.

Examining the data from a set of 13,792 addresses on predominantly client networks at the university examined above shows that flow concurrency per client is quite stable

[3]These wildly outlying bins indicate scanning activity or transient bursts; we have not filtered out such activity as, naturally, any flow-state-keeping device must also cope with it.

flow concurrency	median	95^{th}	peak
all (/11)	148k	322k	436k
university (5x/18)	3.2k	4.3k	22.9k

flows/active hosts	median	95^{th}	peak
clients	5.8	11.8	53.8
servers	10.3	13.4	23.0
content	16.5	43.3	49.8
all	5.2	7.7	9.7

Table 1: Summary of results

but potentially long-tailed, due to highly active clients as well as scanning activity. There are 5.8 concurrent flows per active client in the median 5-second bin, 11.8 in the 95^{th}-%ile bin, and 53.8 in the peak bin. Of note is that, in very few bins, is the number of concurrent flows per active host less than 4: the5^{th}-%ile bin already has 3.8 concurrent flows per active host; we interpret this to be an effect, in part, of the dominance of Web traffic in client networks, and the common browser behavior of opening four simultaneous sockets per transaction.

Similarly, for 13,376 addresses on predominantly server networks at the same university, we see higher flow concurrency per internal host, as would be expected: 10.3 concurrent flows per active host median, 13.4 95^{th}-%ile, 23.0 peak.

Turning to content provider networks, we looked at all traffic crossing the border of the measured network from AS 15169 (Google), AS 20490 (Akamai in Europe), and AS 36040 (also Google); these 709 prefixes account for about 1.4 million IP addresses, although only very few of these are seen in our data set: in the peak bin, only 1,254 unique hosts from these prefixes are observed. We assume that all the internal hosts communicating with these networks represent clients, and the distribution of flow concurrencies is consistent with this (5.6 flows per active internal host median, 7.1 95^{th}-%ile) However, we see higher flow concurrency per external (server) host: 16.5 flows per active server median, 43.3 95^{th}-%ile, 49.8 peak. Note that these numbers are biased by the fact that we only observe traffic crossing our border, so represent a *lower* bound on flow concurrency for these networks.

At higher levels of aggregation, active hosts are balanced out by inactive hosts, transients are amortized over a larger address space, and observed flow concurrency per host goes down: for the entire network in aggregate, we observe 5.2 flows per active internal host median, 7.7 95^{th}-%ile, 9.7 peak.

From these findings we determine that client concurrency per active host is relatively stable, while server concurrency per active host is (1) generally higher than client concurrency and (2) highly dependent on server popularity.

3.1 The effect of short flows

Figure 2 shows the cumulative distribution of the durations of 150,000 flows uniformly sampled from the data set. As seen here, most flows are short: the median flow duration is just 256ms, and 30.7% of flows have a duration of 0ms[4].

[4]As noted in [16], flow durations in our data set are quantized to 64ms by the implementation of the metering process; these are therefore single-packet flows (23.0%), or durations less than the minimum timing resolution.

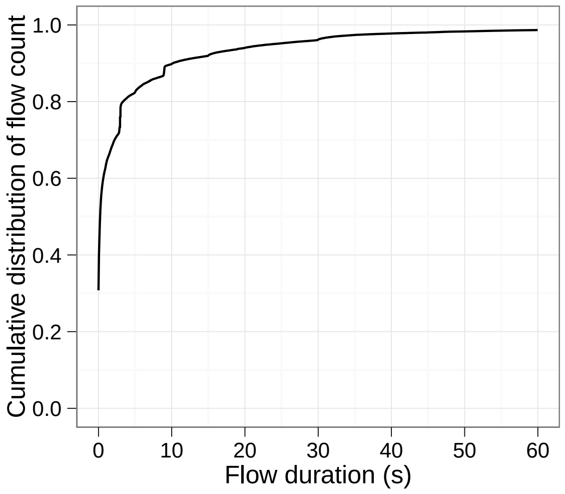

Figure 2: Flow count by duration

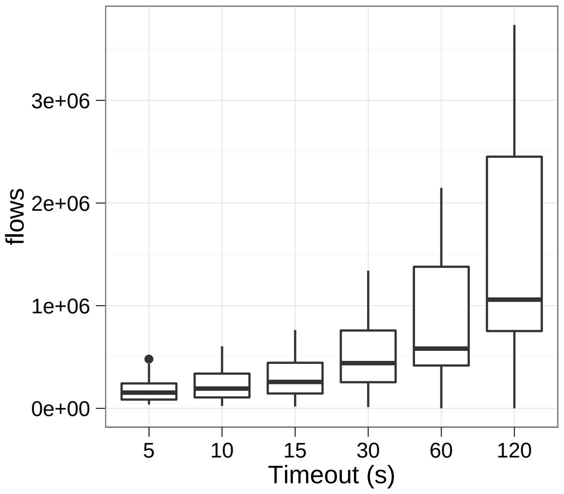

Figure 3: Flow concurrency by timeout

These short flows account for very little total traffic volume but account for the majority of concurrent flows at any given instant: flows with duration 256ms or less account for 51.7% of flows, but only 8.34% of packets and 5.61% of bytes in the examined traffic sample. Much of this short-flow traffic is related to scanning or other nonproductive activity.

This has important implications for flow concurrency. First, it means that observed flow concurrency is highly dependent on the chosen t, as most flows are short. Flow keys for client-generated traffic rarely repeat in the short term due to ephemeral port selection, so newly-initiated flows almost always take a new entry in the flow table. This effect can be seen in figure 3, which shows flow concurrency over a single hour of the data for multiple values of t.

Examining 95^{th}-%ile concurrency, we find that while doubling t from 5s to 10s increases observed by about a third, from 301,000 to 407,000 flows, doubling t from 15s to 30s nearly doubles it, from 534,000 to 912,000.

As each subsequent doubling of t over 15s shows a nearly linear doubling of observed concurrency, the numbers in the findings above can be roughly linearly scaled to generate flow concurrency estimates for different timeouts. at $t = 30s$, for instance, the rule of thumb derived from the measured network would indicate a peak flow concurrency of about 70,000 per /16.

A second effect of short flows can be seen in the correlation of flow concurrency with traffic volume. Obviously, since each flow contributes to the total traffic volume, concurrency and volume are positively correlated. However, since large flows contribute more to total volume than to flow count, the two metrics weigh short and long flows differently. This can be seen in the correlation between the two, which is 0.668 with $t = 5$.

3.2 Long-term trends

To this point we have presented a picture of the state of the network as of 2011; we look at how this trend has developed over time to predict future developments. Here, we look at comparable weekday time spans and traffic across the entire network in 2003, 2005, 2007, and 2009, and compare these to 2011.

We find that flow concurrency does indeed grow along with general traffic volume over the 8-year period examined, as shown in figure 4. This indicates a need to continue scaling flow state with traffic growth. However, we do not see a strong long-term trend in concurrent flows per active internal host on the network: 95^{th}-%ile numbers increase slightly from 5.90 in 2003 to 7.74 in 2011. This effect can be explained both by changing usage patterns on the measured network over the 8-year period as well as a potential slight increase in overall flow concurrency per host. However, we see no compelling evidence to conclude that change in network protocols and applications has led to an increase in flow concurrency at this level of aggregation.

4. DISCUSSION

These findings on flow concurrency apply to capacity sharing in two ways: with respect to scalability and deployability of audit devices used to enforce sharing, and with respect to flow state in the Internet itself as a congestible resource; we discuss these in this section.

4.1 Scalability of audit devices

Moving audit and drop to the network edge, as in conex [12], greatly increases scalability of these functions. Assuming a peak load around 12 concurrent flows per active client, and on the order of 64 bytes per flow of required storage (keys, mark counters, and additional statistics), each client /24 requires a maximum of 200kB of state, in the case that every host is active all the time; an average /24 on the observed network would require only 50kB.

Concurrency on server networks, however, greatly depends on the server's popularity, as shown in section 3. It is therefore more difficult to provide guidance for server networks. However, even assuming two orders of magnitude higher concurrency at servers, a conservative estimate based on our data, 20MB of state per /24 will not impose particularly difficult requirements on audit devices designed for content

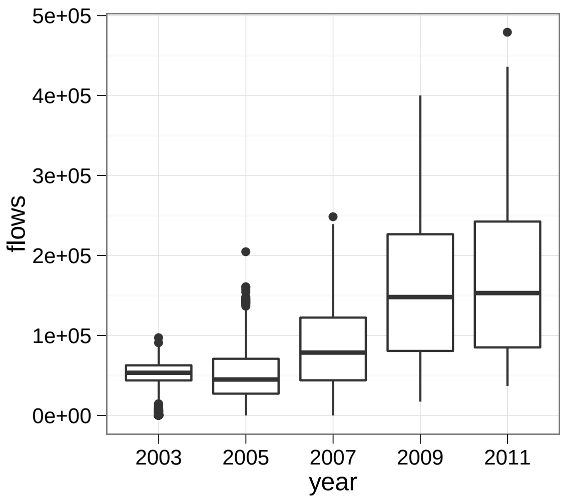

Figure 4: Flow concurrency by year

provider networks. These state requirements can of course be reduced by adding graceful degradation such as deterministic sampling to handle peaks in flow concurrency, at the cost of audit fidelity.

In any case, the relationship between observed concurrency (and therefore flow state requirements) and timeout t shown in section 3.1 would give clear guidance to build audit devices that aggressively time out idle flows, with timeouts on the order of 15 seconds or less, in cases in which flow state may be a limited resource.

4.2 Flow-state congestion

While auditing has the luxury of edge deployment and the possibility of graceful degradation to deal with high flow concurrency, many applications requiring flow state do not, e.g. NAT devices for IPv6 transition. These, by design, must be deployed at large interconnection points, and must deal with peak load, or will cause entire flows to drop. While it is to be hoped that such devices will only be temporarily deployed, it is possible that transition to IPv6 will require the use of address translation in the network for multiple decades.

Given long-term trends in flow concurrency, and the unsustainability of over-provisioning of flow tables in flow-state-keeping devices, we foresee that flow state within these devices may become a resource which may become congested, and for which fair allocation is necessary, just as space in queues.

As discussed in section 1, this problem is indeed addressed in the literature, focusing on fair allocation of resources on flow-state-keeping devices in order to increase the load these devices can handle. In essence, the aim of this work is to keep up with increasing middlebox deployment by continuing to increase the packet rates and flow concurrencies the devices can handle. Improvements in scheduling as in [9], which take state requirements into account in scheduling, build on prior improvements, e.g [8]. As the only degree of freedom these devices have is the order in which they service incoming packets, this work all builds in some way on fair queueing [6].

In addressing flow-state fairness through scheduling, the only way to reduce flow concurrency at a given point is temporal offload: delaying or rejecting packets that would necessitate new entries in the flow table (e.g. TCP SYN packets). As shown in section 3, peak concurrency is generally much higher than median concurrency, so most of the state in such devices is transient. As shown in section 3.1, most flows are short, so entries may be aggressively timed out. Therefore a strategy based on delaying SYNs for space in the flow table may indeed allow middleboxes to keep up with transient peaks above their nominal flow table capacity.

However, in the larger sense, this simply moves the problem around. A delayed SYN which cannot be processed within a short period of time must be dropped. A dropped SYN, or a SYN delayed too long, will lead to retransmission by the connection initiator, which in turn increases non-productive traffic in the network. Delays and drops are also visible at the application layer, leading to long-user visible delays, and are indistinguishable from a link or host outage condition.

Considering flow state congestion across multiple devices along a path complicates this, as flow state congestion is more likely to occur closer to the network core, where higher degrees of aggregation lead to greater concurrency. This is counter to the common assumption that the access link is the bottleneck link. Of course, coming closer to the core of the network one encounters fewer and fewer devices keeping state per flow, because it is not necessary for routing decisions. However, the proliferation of middleboxes and deployment of "carrier-grade" NAT begins to call this assumption into question. State exhaustion closer to the core will cause wasted flow state in devices earlier in the path, as they will have already set up state for the flow on the initial SYN.

Approaches for flow reservation on TCP/IP networks are impractical at the end-system level precisely given the dominance of short flows. Indeed, all such approaches (e.g. RSVP-TE[1]) operate at higher levels of aggregation, and in the cases in which they are applied, the reservations are used for relatively small groups of long-duration, high-volume flows.

On a yet larger scale, spatial offload of flow state could be possible out-of-band via routing table manipulation: changing intra-domain routes in order to balance state from over-utilized devices to under-utilized devices in advance of state exhaustion. However, this would introduce yet another constraint on routing, which is already a difficult problem, and has convergence times much longer than most transient peaks in flow concurrency. Further, intra-domain routing changes may impact inter-domain routing, with implications for global routing stability. We therefore see this as a tool too blunt to be useful for this application.

Finding no satisfactory approach to flow-state offload, we turn to approaches to discourage flow state congestion, by analogy with capacity sharing approaches. Current in-band signaling to discourage queue congestion is not well-suited to addressing flow state congestion, as it requires two-way communication to be established for signaling to begin, i.e., after flows are already taking up state in the network.

Given the relative dominance of short flows, a signaling method in which senders declare the planned duration of a flow in advance would allow devices in the network to plan flow table occupancy in the short term, and to give different incentives to short and long flows. We note that the information required to make this work is often available at the application layer and can be heuristically determined ay lower layers; we further note that application-layer approaches for performance enhancement which use multiplexing (e.g. Google's SPDY [2]) show that it is possible to reduce flow concurrency due to a given end system. By analogy with queue capacity sharing, this facility would require an audit function to verify the honest participation of senders.

We present this as a point for discussion.

4.3 Conclusions

In this work, we have presented a simple methodology for measuring flow concurrency from network flow data, which can be used to estimate flow state requirements for flow-state-keeping devices in the network. We apply this to flow data captured from a national-scale network, and find that flow concurrency is generally positively correlated with traffic volume, and stable on a per-active-host basis depending on the aggregate activity of the hosts on a network. We additionally find that flow concurrency is dominated by short flows, which indicates the use of aggressive timeouts on flow-state-keeping devices.

Flow state requirements derived from flow concurrency measurements on a production network confirm the architectural assumption that as long as audit devices for capacity sharing are deployed at the network edge, flow state requirements are not the limiting factor in their deployment.

However, increasing use of flow-state-keeping devices suggests that fair allocation of flow state, as queue space, may become necessary in the future.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge SWITCH for providing the data used in this study, and Simon Leinen for discussions on the subject. Thanks to Elias Raftopoulos for his assistance in identifying clients and servers, and to Stephan Neuhaus for his comments. This work is partially supported by the European Commission under grant agreement FP7-318627 (mPlane).

6. REFERENCES

[1] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), Dec. 2001.

[2] M. Belshe and R. Peon. SPDY Protocol. IETF Internet-Draft (work in progress): `http://tools.ietf.org/pdf/draft-mbelshe-httpbis-spdy-00.pdf`.

[3] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho. Seven Years and One Day: Sketching the Evolution of Internet Traffic. In *Proc. IEEE INFOCOM*, 2009.

[4] B. Briscoe, A. Jacquet, C. D. Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing congestion response in an internetwork using re-feedback. *ACM SIGCOMM CCR*, 35(4):277–288, Aug. 2005.

[5] B. Briscoe, R. Woundy, and A. Cooper. Congestion Exposure (ConEx) Concepts and Abstract Mechanism, July 2012. IETF Internet-Draft (work in progress): `http://tools.ietf.org/pdf/draft-ietf-conex-concepts-uses-05.pdf`.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19(4):1–12, Sept. 1989.

[7] C. Donley, L. Howard, V. Kuarsingh, and J. Berg. Assessing the Impact of Carrier-Grade NAT on Network Applications, October 2012. IETF Internet-Draft (work in progress): `http://tools.ietf.org/pdf/draft-donley-nat444-impacts-05.pdf`.

[8] N. Egi, A. Greenhalgh, M. Handley, G. Iannaccone, M. Manesh, L. Mathy, and S. Ratnasamy. Improved forwarding architecture and resource management for multi-core software routers. In *Proceedings of the 2009 Sixth IFIP International Conference on Network and Parallel Computing*, NPC '09, pages 117–124, 2009.

[9] A. Ghosdi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. In *Proc. of ACM SIGCOMM*, 2012.

[10] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proc. ACM Internet Measurement Conference*, pages 181–194, Berlin, Germany, 2011.

[11] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet Inter-Domain Traffic. In *Proc. of ACM SIGCOMM*, 2010.

[12] M. Mathis and B. Briscoe. Congestion Exposure (ConEx) Concepts and Abstract Mechanism, July 2012. IETF Internet-Draft (work in progress): `http://tools.ietf.org/pdf/draft-ietf-conex-abstract-mech-05.pdf`.

[13] D. Schatzmann, S. Leinen, J. Kögel, and W. Mühlbauer. FACT: Flow-based approach for connectivity tracking. In *Proc. Passive and Active Measurement Conference*, 2011.

[14] B. Tellenbach, M. Burkhart, D. Schatzmann, D. Gugelmann, and D. Sornette. Accurate network anomaly classification with generalized entropy metrics. *Computer Networks*, 55(15):3485–3502, October 2011.

[15] B. Trammell, B. Claise, and A. Wagner. Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol, August 2012. IETF Internet-Draft (work in progress): `http://tools.ietf.org/pdf/draft-ietf-ipfix-a9n-06.pdf`.

[16] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart. Peeling away timing error in NetFlow data. In *Proc. Passive and Active Measurement Conference*, 2011.