Rethinking Networking for "Five Computers"

Sundararajan Renganathan
Microsoft Research India
t-sur@microsoft.com

Venkata N. Padmanabhan
Microsoft Research India
padmanab@microsoft.com

Akshay Uttama Nambi
Microsoft Research India
t-snaksh@microsoft.com

ABSTRACT

T. J. Watson's apocryphal statement about there being a market for only "five computers" has, in a sense, come true with the rise of cloud computing and the dominance of a handful of "mega-computers" in terms of Internet traffic volume. However, network protocols and operation over the Internet have, for the most part, remained wedded to the old world, with individual hosts operating autonomously. We argue that this is suboptimal and that the time has come to revisit networking in the world of "five computers." We consider various networking functions, including specifically congestion control and network diagnosis, and provide an indication of the potential benefits of a new coordinated approach and sketch out an approach to realizing these benefits.

1 INTRODUCTION

It has been suggested that T. J. Watson of IBM had, in 1943, made the statement, "I think there is a world market for maybe five computers." [8, 9]. Although it is quite likely that he never made this statement [3], it is often held up as an example of a technology prediction that has proved to be grossly off the mark. However, in a sense, this "prediction" has nonetheless come to be true with the rise of cloud computing [19, 34]. A handful of large cloud providers and services account for the majority of Internet traffic. For instance, with the rise in popularity of online video, Netflix and Youtube alone reportedly account for over 50% of Internet traffic [5, 7]. Even traditional "peer-to-peer" applications such as A/V conferencing (e.g., Skype) have, for reasons of connectivity and performance, moved to an architecture where communication happens between clients and cloud-based relays. Therefore, the vast majority of Internet traffic has at least one end-point on one of "five" computers (of course, we mean "five" not in a literal sense but rather figuratively to mean a "handful").

Internet protocols such as TCP were designed for a decentralized world of host-to-host communication. Hosts make minimal assumptions about the network and operate largely autonomously. For instance, when a new TCP connection is launched, it generally starts with a clean slate and goes through its motions in accordance with the parameters (e.g., initial congestion window size and the slow-start threshold) that it has been configured with. Even where it uses history [27] and/or machine learning [20, 45] to adapt, it is generally limited to its local view. Likewise, when there is a failure, the host notices it only when a communication attempt fails, and then has to rely on local tools such as `ping` and `traceroute` to diagnose the problem. The local view also means that the network API is largely bereft of predictive information, e.g., how long a download is likely to take or how good the quality of a video call is likely to be.

Clearly, there is the opportunity to do better individually and collectively through information sharing and coordination across senders. We touch on a couple of examples here.

- If the shared information indicates that the network is congested (e.g., a significant packet loss rate), it would be advantageous for the network as a whole if each sender were to be less aggressive. In the context of TCP, that might mean using a small initial window or backing off more sharply in the event of a packet loss.
- Shared information can also help detect and diagnose network problems, across senders or even across services. For instance, if users of a cloud provider are experiencing unreliability in their VoIP service but not their file hosting service, that would point to a VoIP-specific issue.

While there has been prior work on creating an "information plane" for Internet hosts [32], we argue that the present "five-computer" world presents a new opportunity for rethinking and redesign that has hitherto not been available.

Let us consider an example. Netflix has been and continues to be dominant in terms of its share of Internet traffic (e.g., it accounted for 37% of Internet traffic in 2015 [5]). However, it only has a relatively modest number of servers pumping out this traffic (e.g., a research effort from 2016 aimed at mapping Netflix's servers only found 4669 servers [4])[1]. Sharing information across and coordinating a few thousand servers run by a single entity, is likely to be much more *feasible* than it would be across millions or billions of disparate end-points or peers. And where such an entity accounts for a dominant share of Internet traffic, such information sharing and coordination is also likely to be *effective*. Furthermore, we argue that even if

[1] Although Netflix has moved many of their services to Amazon AWS, the actual delivery of media bits, which arguably accounts for the bulk of their traffic, happens from servers in the Netflix OpenConnect CDN [2, 6]

HotNets-XVII, November 15–16, 2018, Redmond, WA, USA

ACM ISBN 978-1-4503-6120-0/18/11…$15.00
https://doi.org/10.1145/3286062.3286076

such coordination is confined to the individual entities among the "five computers" and does not span multiple competing entities (e.g., Netflix, Youtube, and Amazon), there would still be tangible benefits since the large scale of the individual entities would help push the network towards a more efficient operating point.

In the rest of the paper, we present our proposal for information sharing and coordination, which we dub as `Phi`, a play on the word "five".

2 RETHINKING CONGESTION CONTROL

Congestion control is key to the stability and efficient operation of networks. There have been myriad flavors of congestion control schemes over the years, varying in the control mechanism employed (e.g., window-based or rate-based), feedback relied upon (e.g., delay, loss, explicit congestion notification), the control policy (e.g., the increase or decrease policy), and more [11, 14, 27, 30, 37, 38]. In the recent years, there has been recognition of the fact that a static and handcrafted control policy is unlikely to be optimal, because these policies are often based on a simplified model of the network that might not reflect reality. This has led to work on machine-learned congestion control, where the congestion control algorithm is trained offline using trace-driven simulation and uses the resultant model to respond to a client's measurement of end-to-end metrics such as delay and packet loss [43, 45]. Nevertheless, these approaches still have clients operating autonomously, each making its own measurements and responding according to the model.

We argue that in a "five-computer" world, there is the opportunity to do much better, in terms of both information sharing and coordination. After all, if a single entity such as Netflix accounts for a dominant share of network traffic, then it makes little sense for new streams to fly blind, oblivious to what Netflix already knows about the network. For instance, if the experience from ongoing streams points to a high level of congestion on the network path to Comcast customers in a particular location, it would likely be beneficial for a new stream to learn from that and temper its congestion control accordingly. Furthermore, since the prevalence of FIFO queueing makes the network not incentive compatible [23], there would need to be a degree of coordination across the streams to ensure overall benefit. While the "five-computer" world, with the dominance of just a small number of entities, facilitates such coordination, making coordination practical is still a key challenge.

In the remainder of this section, we expand on the opportunity for and the benefits of information sharing and coordination in the "five-computer" world in the context of congestion control, and outline a practical design for realizing such sharing and coordination.

2.1 Opportunity for Sharing

We look at Internet-bound egress traffic from a large cloud provider to assess the opportunity for the sharing of information. In particular, we use the IPFIX (IP Flow Information Export) data [18] to calculate the number of TCP flows (characterized by the number of unique 4-tuples <Src Ip, Src Port, Dst Ip, Dst Port>) per minute for each /24 subnet that the provider sends traffic to. Given this compact spatio-temporal granularity (/24 subnet and 1-minute time slice), we can reasonably expect all the flows to follow the same WAN path.

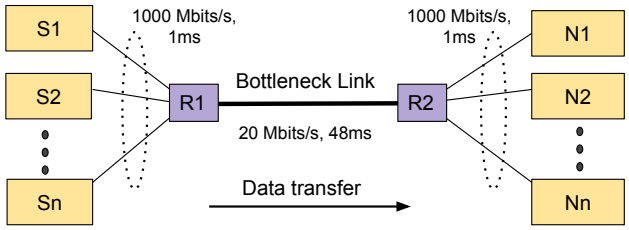


Figure 1: Dumbbell network used for the TCP Cubic experiments. The buffer size is 5 times the bandwidth-delay product of the bottleneck link.

The IPFIX sampling rate is set to 4096 at each router meaning that one in 4096 packets traversing the router is sampled and the headers of these sampled packets are reported to the centralized collector service. Despite this aggressive subsampling, we find that 50% of the flows share the WAN path with at least 5 other flows while 12% share it with at least 100 other flows. The actual sharing (without the sub-sampling) is likely to be much higher.

Another pertinent question is whether these flows share a bottleneck link. Recent work on egress path selection [41, 46] suggests that the bottleneck link is often *not* the last hop. Nevertheless, a measurement study with techniques such as [29] would be needed to establish whether a set of flows share a bottleneck link.

2.2 Benefits of Sharing and Coordination

We first consider a case where there is both sharing of information and coordination across senders. To make our discussion concrete, we consider the simulated network topology depicted in Figure 1, with a single bottleneck link. We use a varying number of TCP senders in ns-2 (version 2.35) to transmit data across this bottleneck link. Furthermore, each sender launches fresh connections sequentially ("on" periods) separated by idle "off" periods, where the amount of data transferred during "on" periods and the duration of "off" periods are picked from separate exponential distributions. The varying workload generates different levels of congestion at the bottleneck link, with average link utilization spanning from 20% to 80% across the experiments.

2.2.1 TCP Cubic. We first consider TCP Cubic, which is widely used and is the default flavor of TCP in Linux and recently also in Windows. Our goal is to compare the performance of TCP Cubic, with its parameters tuned for the current conditions based on globally shared knowledge, with that of TCP Cubic with its default (and fixed) parameter settings. We start by considering 3 different parameters: `windowInit_` (initial congestion window), `initial_ssthresh` (initial slow start threshold), and β (where (1-β) is the multiplicative decrease factor applied on packet loss). The default settings

Parameter	Default Value
`initial_ssthresh`	Arbitrarily large (65K segments)
`windowInit_`	2 segments
β	0.2

Table 1: Default settings of the TCP Cubic parameters.

Parameter	Range	Increment
`initial_ssthresh`	2 - 256 segments	$\times 2$
`windowInit_`	2 - 256 segments	$\times 2$
β	0.1 - 0.9	+ 0.1

Table 2: Range of parameter sweep in TCP Cubic-`Phi`

of these parameters in the TCP Cubic implementation in ns-2 are given in Table 1. Note that to enable effective bandwidth discovery, RFC-5681 recommends that the initial slow start threshold be set arbitrarily large. [2]

For each level of workload, we perform a sweep across the range of parameter values noted in Table 2 and identify the optimal parameter setting. To define optimality in our experiments, we start with the network power metric [22], $P = \frac{r}{d}$, where r is the throughput or data rate, and d is the delay, and extend it to also incorporate the packet loss rate, l (inspired by [31]), yielding the new metric, $P_l = \frac{r(1-l)}{d}$. We use P_l is the metric to optimize in the case of TCP Cubic and $log(P)$ in the case of Remy (in line with [45]).

We first consider a simplified setting, where it is assumed that for a given run, all the TCP Cubic senders use the same parameter settings that is fixed for the duration of the run. (We defer to Section 2.2.2 discussion of a more realistic setting, wherein the parameters are set on the fly as connections come.) The workload is varied by varying the number of senders that share the bottleneck link, the average connection length for "on" periods and the average duration of the "off" periods. For each combination of workload level and parameter settings, we repeat the experiment for $n = 8$ runs.

Figure 2a and 2b show the throughput and queueing delay corresponding to the different workload levels, where the average connection length is set to 500 KB and the average "off" time is set to 2 seconds. (Note that the throughput is computed only during the on-times, i.e., throughput = bits transferred / ontime.) The aggregate throughput and delay measurements are averaged across the corresponding set of runs. In each case, the solid triangle marker corresponds to the default parameter settings for TCP Cubic, the circles to other parameter settings, and the solid circle to the optimal setting. The size of each marker indicates the packet loss rate; the larger the size, the higher the loss rate.

The optimal parameter setting yields a significantly higher throughput and lower queueing delay than the default setting. It also achieves a lower packet loss rate (e.g., 0.01% vs. 3.92% in the case shown in Figure 2b). The optimal case uses a larger initial window but a smaller slow start threshold than the default case. And as we would expect, the optimal settings of these parameters shift to be smaller as the link utilization becomes higher. Finally, in these settings, modifying β does not have an impact because each connection tends to be relatively short.

Is the improved performance merely a statistical fluke or is it that the non-default parameter settings provide benefit consistently? To answer this question, we perform leave-out-one validation, wherein for each workload, we take the "optimal" parameter settings from one run and evaluate its performance on the remaining $n - 1 = 7$ runs. As shown in Figure 3, applying such a common parameter setting to all runs yields significant performance gains over the default setting, almost equal to the gains from the "optimal" setting for each run. (Note that we are optimizing for the P_l metric, so it is possible for the "common" setting to be better than the "optimal" setting on a subset of the individual metrics, e.g., throughput or delay.) This stability across the runs for a particular workload shows that the gains are not a fluke.

Figure 2c shows the results with 100 long-running connections, with the bottleneck link being 99%. Unsurprisingly, in this setting, varying the initial window size or the slow start threshold does not have much impact. However, β does have a significant impact, with a larger value (corresponding to a sharper back-off upon packet loss) yielding a significantly lower queueing delay compared to the default.

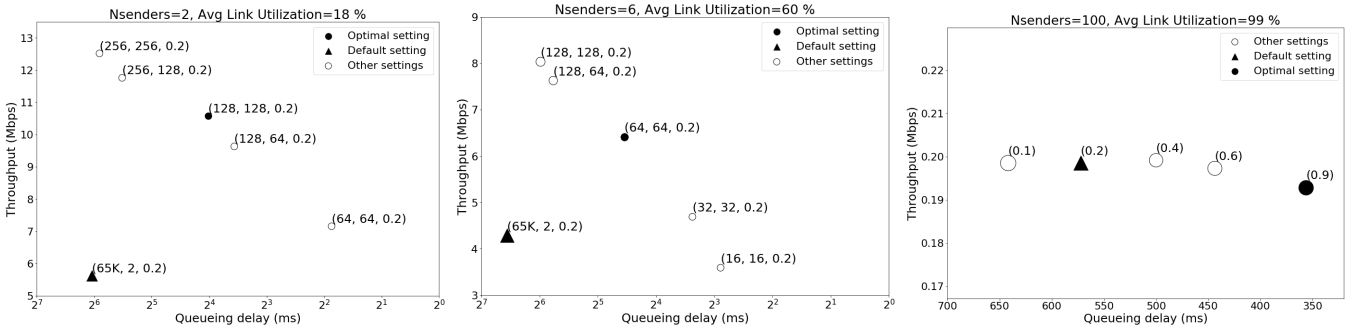
Since the optimal parameter setting depends on the level of congestion, a natural question is how `Phi` senders could assess the network conditions and set the TCP Cubic parameters accordingly. We turn to this next.

2.2.2 Practical Approach to Realization. As discussed above, the optimal parameter setting depends on the level of congestion. We argue that the *congestion context* can be characterized in terms of (i) the utilization of the bottleneck link (u), (ii) the queue occupancy (q), and (iii) the number of competing senders (n). For instance, when any of these metrics is high, that would mean a high level for congestion and would call for more conservative behavior, and conversely a low level of congestion would mean that less conservative behavior would still be safe.

This then brings up the question of how the congestion context could be estimated. To this end, we envisage a *context server*, say within a domain (i.e., within one of the "five" computers), that serves as the repository of shared state from which the congestion context can be computed. Information from senders on when and how much data is transferred would enable estimation of u and n, while the difference between the current RTT and the minimum RTT would give an indication of q (as in [45]).

While instantaneous sharing of state by the individual senders might be ideal, it would clearly be prohibitive in terms of the overhead and would limit scaling. So we keep the communication between the senders and the context server minimal — each sender would look up the context server once when a new connection starts (so that it can then determine the optimal parameter settings) and would report back to the context server once the connection ends (so that the shared

[2]RFC-5681 [10] recommends that "*The initial value of ssthresh SHOULD be set arbitrarily high (e.g., to the size of the largest possible advertised window)*". In our experiments, we set it to 65536, or 65K, segments.



(a) Low Link Utilization **(b) High Link Utilization** **(c) Long running connections.**

Figure 2: Cubic parameters (`initial_ssthresh`, `windowInit_`, β) for various workloads. Note that only the parameter (β) has been reported for sub-figure (c).

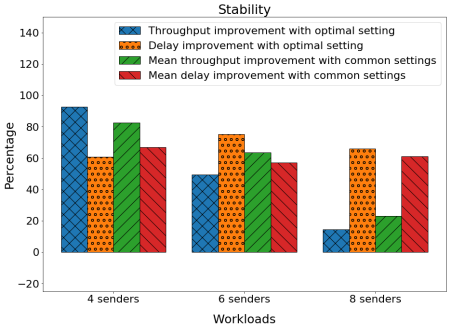


Figure 3: Stability analysis of improvement with optimal parameter setting, over the default parameter setting.

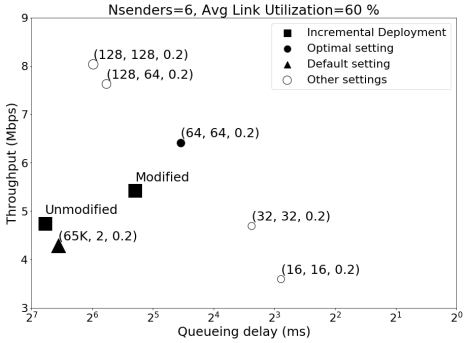


Figure 4: Cubic parameters for incremental deployment

state can be updated based on the experience of that connection). As we show in Section 2.2.4, such a practical approach, with minimal overhead, still provides significant gains.

Note that we use connections as the unit of sharing and coordination only as a matter of convenience. There is nothing that requires it to be so. If connections are short, we could look up and report back to the context server only once in many connections. Likewise, if the connections are long, we could communicate with the context server multiple times within the same connection. The goal is just to ensure that sufficiently up-to-date information on the state of the network (the "network weather") is available to individual hosts for them decide on suitable settings for current conditions.

2.2.3 Incremental Deployment. Thus far, we have only considered a cooperative scenario, where all senders share information and conform to the optimal parameter setting. However, since transitioning to the proposed approach of sharing and coordination is likely to be gradual, the question is whether a partial deployment would also offer any benefit.

To evaluate this, we consider a setting where one half of the senders ("unmodified") sticks with the default parameter settings for TCP Cubic, while the other half ("modified") uses the parameter setting that would have been optimal had all senders been cooperating. As we see in Figure 4, the modified senders still see improved throughput and delay compared to the default case. Even the unmodified senders see an improvement in the power metric, though the queueing delay is slightly worse.

As noted in Section 2.2.1, unmodified senders start with a large default slow start threshold (65K segments) for each connection, i.e., each "on" period). In contrast, modified senders start with a slow start threshold of 64 segments found to be optimal for the current network conditions. Thus, the unmodified senders tend to fill up the queue, even causing packet loss (as can be seen from the size of the markers), much more than the modified senders do. Given FIFO queueing, this would affect all flows. However, the moderate link utilization (60%) means that modified flows sometimes get lucky in not encountering any unmodified flows (i.e., all the unmodified senders happen to be in the "off" state), thereby enjoying a lower delay. As the utilization goes higher, though, any advantage enjoyed by the modified flows diminishes.

Thus, even a partial deployment of `Phi` can offer benefits in certain circumstances. We discuss this further in Section 3 below.

2.2.4 Remy. In the recent years, there has been a growing body of work on a machine learning based approach to congestion control, with Remy [45] perhaps being the earliest such example. The general idea is to move away from hand-crafted and hard-coded congestion control strategies (as, for example, in TCP Cubic) and instead to learn the appropriate congestion control response in any given situation based on training experiments typically done in a simulator.

The question we ask is whether the proposed sharing and coordination in `Phi` would provide any benefits over a scheme such as Remy that is already adaptive. The reason it might is that sharing of information would enable senders to obtain a

Algorithm	Median throughput (Mbps)	Median queueing delay (ms)	Median objective function
Remy-Phi-practical	1.93	5.6	2.52
Remy-Phi-ideal	1.97	3.0	2.56
Remy	1.45	1.7	2.26
Cubic	1.03	9.3	1.87

Table 3: Results for single bottleneck dumbbell topology with link speed 15 Mbps and round-trip time 150 ms with 8 senders, each alternating between flows of exponentially-distributed byte length (mean 100 KB) and exponentially-distributed off time (mean 0.5 s)

more accurate picture of the network weather and do so more quickly than they would when operating individually as in Remy. For instance, if the network is under heavy congestion, the senders in Phi could directly learn of this (e.g., based on shared information on bottleneck bandwidth utilization) and adopt a suitably conservative congestion response instead of each sender having to discover it individually, and hence slowly, based on its own measurements.

To investigate this question, we extend the context (or "memory" in Remy parlance) maintained by each Remy sender with an additional dimension corresponding to the bottleneck link utilization, u. We then retrain Remy in the same range of network and traffic model parameters as reported in the original Remy paper [45]. Note that, during training, we allow each sender access to up-to-the-minute link utilization. Table 3 summarizes the results for one of the topologies reported in the original paper, evaluated in ns-2. Remy-Phi-ideal corresponds to the ideal setting where all senders have access to up-to-the-minute bottleneck utilization while in Remy-Phi-practical, the utilization information is queried at the beginning of a connection and updated at the end of the connection (as outlined in Section 2.2.2 above). From Table 3, we see that Remy-Phi-ideal yields a significantly higher throughput than unmodified Remy, and while it also results in a marginal increase in the queueing delay, Remy's objective function, $log(P)$, is nevertheless higher (because the marginal increase in queueing delay does not have much of a negative impact on overall delay). Remy-Phi-practical performs somewhat worse than Remy-Phi-ideal but still better than Remy.

Thus, we believe Phi could be beneficial even in the context of modern machine learning based congestion control.

3 OTHER ISSUES AND OPPORTUNITIES

3.1 Incentives for Adoption

As noted in Section 2.2.3, unless we are considering a situation with low link utilization, benefiting from Phi would require cooperation from the majority of or all senders. The reason is that the prevalence of FIFO queuing means that a flow is not insulated from the actions of other flows. Indeed, prior work has pointed out that FIFO queuing is not incentives-compatible [23]. Even so, TCP congestion control and TCP friendliness has taken root through a process of techno-social "(dis)incentives", through forums such as the IETF, and the Internet has not suffered a congestion collapse episode since the 1980s.

We believe that a similar combination of technical and social pressures could spur the adoption of Phi. Large operators would be expected to adopt Phi because of the beneficial impact on themselves and on the network at large. The information to be shared between providers, to establish a common barometer on the network weather, would be minimal (e.g. the level of congestion in a particular part of the network). Work on secure multiparty computation and anonymous aggregation [15, 39, 40] could be leveraged to further shield such information sharing.

Even if data-related sensitivities prove to be an impediment to the deployment of Phi across mutually-competing entities, the scale of each of the "five computers" (e.g., Netflix) is such that it is likely to be beneficial, even if not optimal, for each entity to employ Phi independently based on its own information. Even if the paths between the content sources and their clients are short, as noted in [17], flows traversing the same bottleneck links on these short paths stand to benefit from Phi. Furthermore, besides deploying Phi on user-facing networks, we believe that large providers can also fruitfully deploy Phi on their inter-DC WANs. Prior work [26, 28], focused on coarse-grained bandwidth allocation on such networks, does not eliminate congestion or packet loss. Therefore, we believe that informed adaptation of transmission rates on the basis of level of congestion is likely to be beneficial in such single-provider settings too.

3.2 Benefits of Sharing without Cooperation

We consider how sharing of information, say within a relatively small subset of senders, could be beneficial to these senders even if the majority of senders do not cooperate. As noted above, FIFO queuing in this context would mean that the congestion state of the network would not really benefit from Phi. So the opportunity for the minority of clients would be in terms of *informed adaptation* based on shared information. For instance, the jitter buffer size for audio-video streaming could be initialized and updated over time based on the shared information. As another example, the threshold of 3 duplicate ACKs typically used to trigger TCP fast retransmission could be adjusted if the experience of other connections suggests that reordering is prevalent.

3.3 Prioritization Across Flows

In the "five-computer" world, a single entity might have a large number of flows that traverse the same bottleneck link [9] even if these are destined to different clients (e.g., see Section 2.1). However, some flows might be more important than others (e.g., an HD movie stream vs. a TCP bulk transfer). In today's setting, with senders operating autonomously, each sender is individually expected to be TCP-friendly. However, in the "five-computer" world, a single entity could have some of its flows be more (or less) aggressive than others (say based on their "importance"), while still ensuring that the ensemble of flows remains TCP-friendly. This is akin to past proposals such as TCP Session [35] and the Congestion Manager [13] except that the prioritization happens across hosts rather than within a single host.

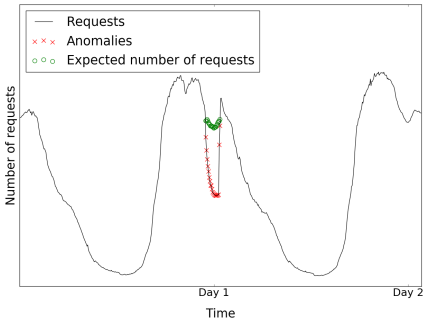


Figure 5: An unreachability event localized to an ISP network in a metro that lasted for around 2 hours.

3.4 Problem Diagnosis

The scale of the "five computers" means that they each have a wide footprint of servers (senders) and clients (receivers), spanning geographical locations, ISPs, and more. Therefore, the sharing of information, even within a single large entity, would provide a diversity of viewpoints to enable far more effective problem diagnosis than would be possible with individual hosts operating autonomously. For instance, when a particular connection faces problems, "comparing notes" with other connections could enable zeroing in on the cause. This is akin to past work on distributed blame attribution [12] except that much of the distributed information in the "five-computer" world is available to individual (large) entities (e.g., Netflix), which makes sharing and diagnosis much more feasible than before.

As a simple but concrete example, consider the problem of network unreachability, wherein a subset of clients is unable to reach a cloud service. Today, individual clients, or users, are left with manually-driven processes such as Down Detector [1] and even social media channels such as Twitter While we could automate the sharing of information across the (disparate) clients, we argue that in the "five-computer" world, it would be more effective for the cloud service, which is the common entity spanning a large number of clients — both the affected ones and the unaffected ones — to aggregate information and thereby perform diagnosis.

We build a time series model for the volume of requests received by a cloud service, sliced along various dimensions (client AS'es, data center locations, etc.), and look for anomalous departures from the model to detect unreachability events and also perform (coarse) diagnosis. Figure 5 shows an unreachability event detected in the context of a large global-scale cloud provider, that was localized to an ISP network on a particular metro.

3.5 Performance Prediction

One functionality that is largely absent in the Internet, despite research proposals stretching back decades [42], is performance prediction. When an application initiates a network flow, it typically does so without any knowledge of how good, or bad, the performance will be. This arises from the autonomous operation of individual hosts. However, in the "five-computer" world, the large volume of aggregate network performance data available even within a single cloud provider would, we believe, enable effective performance prediction. This would mean that before an application downloads a file or makes a VoIP call or launches a video stream, it would be able to obtain an indication of the expected performance, which could even be surfaced to the user (e.g., if the VoIP quality is expected to be poor, the user might hold off on an important call).

4 RELATED WORK

Congestion control has been the focus of much research in networking over decades [14, 25, 27, 38]. While these approaches have varied in terms of their details, two common characteristics have been (a) operation at the level of individual connections, and (b) a hard-coded policy.

There has been been work aimed at addressing both (a) and (b). With regard to the former, the work has centered on host-level aggregation of congestion control [13, 35]. With regard to the latter, there has been much recent work on machine learning based approaches to congestion control and bandwidth adaptation, again based on local input at a host [20, 33, 43, 45]. Our work builds on these but goes beyond by focusing on network information sharing and coordination across the WAN flows emanating from large providers.

There has also been work on aggregating information across hosts for the purposes of performance prediction, diagnosis, etc. [32, 36]. While this body of work was largely set in the context of decentralized peer-to-peer systems, ours is set in the context of large-scale cloud providers, where control by a single entity arguably makes sharing and coordination more feasible than in a decentralized setting. Besides, the prior work has not focused on congestion control.

Finally, there has been work in the context of software-defined networks (SDNs) on centralizing network state and its management [16, 24]. Furthermore, the full visibility into the network state afforded by SDNs, in confined settings such as a data center network, has been used to tune TCP parameters [21], in particular, the initial window size and the retransmit timeout. Other work [44] has proposed sharing of network congestion information by cellular operators to enable better TCP adaptation. Both of these are quite related to our work. However, we focus on settings where it is not feasible to obtain full visibility into the network state, say because the network operator is unwilling to share information.

5 CONCLUSION

We have made the case that in the world of "five computers", network information sharing and coordination across senders and connections could yield significant benefits in terms of congestion control, problem diagnosis, and more. We have presented a practical approach that enables such sharing and coordination with minimal overhead.

ACKNOWLEDGMENTS

We thank our shepherd, Matteo Varvello, and the anonymous HotNets reviewers for their feedback and suggestions.

REFERENCES

[1] Down Detector. http://downdetector.com/.

[2] How Netflix works. https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b.

[3] IBM FAQ, page 26. http://www-03.ibm.com/ibm/history/documents/pdf/faq.pdf.

[4] Locations of 4,669 Servers in Netflix's Content Delivery Network. https://spectrum.ieee.org/tech-talk/telecom/internet/researchers-map-locations-of-4669-servers-in-netflixs-content-delivery-network.

[5] Netflix boasts 37% share of Internet traffic in North America. https://appleinsider.com/articles/16/01/20/netflix-boasts-37-share-of-internet-traffic-in-north-america-compared-with-3-for-apples-itunes.

[6] Netflix Open Connect. https://openconnect.netflix.com/en/.

[7] Streaming services now account for over 70% of peak traffic. https://venturebeat.com/2015/12/07/streaming-services-now-account-for-over-70-of-peak-traffic-in-north-america-netflix-dominates-with-37/.

[8] Thomas J. Watson Wikipedia. https://en.wikipedia.org/wiki/Thomas_J._Watson.

[9] World of five computers. https://www.lexology.com/library/detail.aspx?g=164a442a-1b90-49e3-895d-4c54bb49ecce.

[10] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, Sep. 2009.

[11] V. Arun and H. Balakrishnan. Copa: Congestion Control Combining Objective Optimization with Window Adjustments. In *NSDI*, 2018.

[12] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the Blame Game Out of Data Centers Operations with NetPoirot. In *ACM SIGCOMM*, 2016.

[13] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *ACM SIGCOMM*, 1999.

[14] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *ACM SIGCOMM*, 1994.

[15] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1(101101), 2010.

[16] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[17] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, E. Katz-Bassett, and R. Govindan. Are we one hop away from a better internet? In *Proceedings of the 2015 Internet Measurement Conference*, pages 523–529. ACM, 2015.

[18] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, Sep. 2013.

[19] J. Dean. The Rise of Cloud Computing Systems. In *SOSP History Day 2015*, page 12. ACM, 2015.

[20] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *NSDI*, 2015.

[21] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali. Rethinking End-to-End Congestion Control in Software-Defined Networks. In *ACM HotNets*, 2012.

[22] A. Giessler, J. Häenle, A. König, and E. Pade. Free buffer allocation - an investigation by simulation. *Computer Networks*, 1(3):191–204, Jul. 1978.

[23] P. Godfrey, M. Schapira, A. Zohar, and S. Shenker. Incentive Compatibility and Dynamics of Congestion Control. *ACM SIGMETRICS Performance Evaluation Review*, 38(1):95–106, 2010.

[24] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.

[25] 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.

[26] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 15–26. ACM, 2013.

[27] V. Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[28] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[29] D. Katabi, I. Bazzi, and X. Yang. A Passive Approach for Detecting Shared Bottlenecks. In *Proceedings Tenth International Conference on Computer Communications and Networks*, Oct. 2001.

[30] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *ACM SIGCOMM*, 2002.

[31] L. Kleinrock. Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications. In *International Conference on Communications*, Jun. 1979.

[32] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.

[33] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*, 2017.

[34] D. C. Marinescu. *Cloud Computing: Theory and Practice*. Morgan Kaufmann, 2017.

[35] V. N. Padmanabhan. *Addressing the Challenges of Web Data Transport*. 1998. PhD thesis, UC Berkeley.

[36] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye. NetProfiler: Profiling Wide-Area Networks Using Peer Cooperation. In *IPTPS*, 2005.

[37] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Sep. 2001.

[38] K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer. *ACM SIGCOMM Computer Communication Review*, 18(4):303–313, 1988.

[39] M. Roughan and Y. Zhang. Privacy-preserving performance measurements. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 329–334. ACM, 2006.

[40] M. Roughan and Y. Zhang. Secure distributed data-mining and its application to large-scale network measurements. *ACM SIGCOMM Computer Communication Review*, 36(1):7–14, 2006.

[41] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *ACM SIGCOMM*, 2017.

[42] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *USENIX Symposium on Internet Technologies and Systems*, pages 1–13, 1997.

[43] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. An experimental study of the learnability of congestion control. In *ACM SIGCOMM*, 2014.

[44] A. Terzis and C. Bentzel. Sharing network state with application endpoints. In *Proceedings of the 2015 Managing Radio Networks in an Encrypted World (MaRNEW) Workshop*, 2015.

[45] K. Winstein and H. Balakrishnan. TCP ex machina: Computer-Generated Congestion Control. In *ACM SIGCOMM*, 2013.

[46] K.-K. Yap and et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *ACM SIGCOMM*, 2017.