Rethinking End-to-End Congestion Control in Software-Defined Networks

Monia Ghobadi, Soheil Hassas Yeganeh, Yashar Ganjali
Department of Computer Science, University of Toronto, Canada
{monia, soheil, yganjali}@cs.toronto.edu

ABSTRACT

TCP is designed to operate in a wide range of networks. Without any knowledge of the underlying network and traffic characteristics, TCP is doomed to continuously increase and decrease its congestion window size to embrace changes in network or traffic. In light of emerging popularity of centrally controlled Software-Defined Networks (SDNs), one might wonder whether we can take advantage of the global network view available at the controller to make faster and more accurate congestion control decisions. In this paper, we identify the need and the underlying requirements for a congestion control adaptation mechanism. To this end, we propose OpenTCP as a TCP adaptation framework that works in SDNs. OpenTCP allows network operators to define rules for tuning TCP as a function of network and traffic conditions. We also present a preliminary implementation of OpenTCP in a ∼4000 node data center.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms

Design, Measurement, Performance, Experimentation

Keywords

TCP, Data Center, Software-Defined Network

1. INTRODUCTION AND MOTIVATION

The Transmission Control Protocol (TCP) [18] is designed to fully utilize the network bandwidth while keeping the entire network stable. TCP's behaviour can be sub-optimal and even erroneous [3, 6–8, 28] mainly for two reasons: First, TCP is expected to operate in a *diverse* set of networks with different characteristics and traffic conditions; making TCP a "Jack of all trades, master of none" protocol. Limiting TCP to a specific network and taking advantage of local characteristics of that network can lead to major performance gains. For instance, DCTCP [3] outperforms TCP in data center networks, even though the results might not be applicable in the Internet. With this mindset, one can adjust TCP (the protocol itself and its parameters) to gain better performance in specific networks (*e.g.,* data centers).

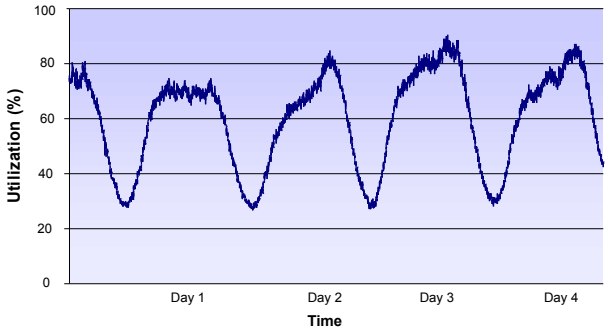

Figure 1: A core link's utilization over time.

Second, even focusing on a particular network, the effect of *dynamic* congestion control adaptation to traffic pattern is not yet well understood in today's networks. Such adaptation can potentially lead to major improvements, as it provides another dimension that today's TCP does not explore.

Example 1 - Dynamic Traffic Patterns: Figure 1 depicts aggregate link utilization of a core link in a backbone service provider in North America. We can see that the link utilization is low for a significant period of time (below 50% for 6–8 hours). The same pattern is seen on all the links in this network. In fact, the presented link has the highest utilization and is considered to be the bottleneck in this network. If the network operator aims at minimizing flow completion times in this network, it makes sense to increase TCP's

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.

initial congestion window size (*init_cwnd*) when the network is not highly utilized (we focus on internal traffic in this example). Ideally, the exact value of *init_cwnd* should be a function of the *network-wide state* (here, the number of flow initiations in the system), and how aggressive the operator wants the system to behave (*congestion control policy*). The operator can define a policy like: if link utilization is below 50%, increase *init_cwnd* to 20 segments. Given the appropriate mechanisms the operator might even dynamically choose the right value for the initial congestion window.

Example 2 - Paced TCP: Let us consider a data center network with small buffer switches. To suppress the burstiness in this network, one may use paced TCP [32]. It has been shown that paced TCP is beneficial only if the total number of concurrent flows are below a certain threshold [15]. In a large data center with thousands to millions of concurrent flows, it is almost impossible to guarantee that the number of active flows will always be less than any threshold. Therefore, providers are reluctant to enable pacing despite its proven effectiveness. However, if we have a system that can measure the number of active flows, we can dynamically switch between paced TCP and regular TCP to ensure the system is always at its peak performance.

Clearly, adapting TCP requires meticulous consideration of the network characteristics and traffic patterns. The fact that TCP relies solely on end-to- end measurements of packet loss or delay as the only sources of feedback from the network means that TCP can, at its best, have a very limited view of the network state (*e.g.,* the trajectory of available bandwidth, congested links, network topology, and traffic). Thus, a natural question here is:

Can we build a system that observes the state and dynamics of a computer network and adapts TCP's behaviour accordingly?

If the answer to this question is positive, it can simplify tuning different TCP parameters (Example 1). It can also facilitate dynamically changing the protocol itself (Example 2).

OpenTCP: This paper takes the first step in addressing the need for a systematic way of adapting TCP to network and traffic conditions. We present OpenTCP as a system for dynamic adaptation of TCP based on network and traffic conditions in Software-Defined Networks (SDNs) [21]. OpenTCP mainly focuses on internal traffic in SDN-based data centers for four reasons: (*i*) the SDN controller already has a global view of the network (such as topology and routing information), (*ii*) the controller can collect any relevant statistics (such as link utilization and traffic matrix), (*iii*) it is straightforward to realize OpenTCP as a control application in SDNs, and (*iv*) the operator can easily customize end-hosts' TCP stack.

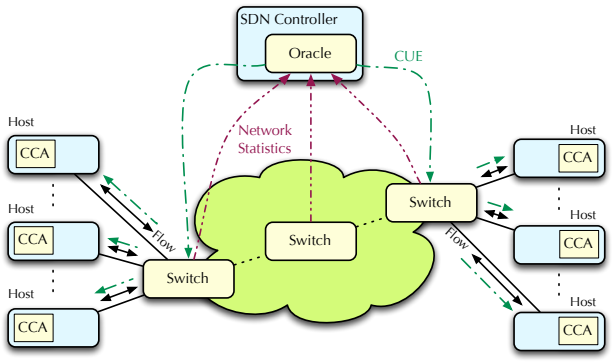

Figure 2: The Oracle, switches, and Congestion Control Agents are the three components of OpenTCP.

Our preliminary implementation of OpenTCP has been deployed in a high-performance computing (HPC) data center with ∼4000 hosts. We use OpenTCP to tune TCP parameters in this environment to show how it can simplify the process of adapting TCP to network and traffic conditions. We also show how modifying TCP using OpenTCP can improve the network performance. Our experiments show that using OpenTCP to adapt *init_cwnd* based on link utilization leads to up to 59% reduction in flow completion times.

OpenTCP and related work: OpenTCP is orthogonal to previous works improving TCP's performance. It is not meant to be a new variation of TCP. Instead, it complements previous efforts by making it easy to switch between different TCP variants automatically (or in a semi-supervised manner), or to tune TCP parameters based on network conditions. For instance, one can use OpenTCP to either utilize DCTCP or CUBIC in a data center environment. The decision on which variant to use is made in advance through the congestion control policies defined by the network operator.

2. OPENTCP ARCHITECTURE

OpenTCP collects data regarding the underlying network state (*e.g.,* topology and routing information) as well as statistics about network traffic (*e.g.,* link utilization and traffic matrix). Then, using this aggregated information and based on *congestion control policies* defined by the network operator, OpenTCP decides on a specific set of adaptations for TCP. Subsequently, OpenTCP sends periodic *Congestion Update Epistles* or CUEs to the end-hosts that, in turn, update their TCP variant using a simple kernel module.

Figure 2 presents the schematic view of how OpenTCP works. As shown, OpenTCP's architecture has three main components. The ***Oracle*** is a SDN controller application that lies at the heart of

OpenTCP. It collects information about the underlying network and traffic. Then, based on congestion control policies defined by the network operator, the Oracle finds the appropriate changes needed to adapt TCP. Finally, it distributes update messages (CUEs) to end-hosts. ***Network switches*** are typical SDN-compatible switches, which in addition to forwarding can collect traffic statistics. A ***Congestion Control Agent*** (***CCA***) is a kernel module installed at each end-host. CCAs receive update messages from the Oracle via the network switches and are responsible for modifying the TCP stack at each host.

Congestion control policies: Although OpenTCP is designed to dynamically adapt TCP, all major decisions are made by the network operator in the form of congestion control policies. Congestion control policies are used to specify which statistics need to be collected, what is the target operational goal (*e.g.,* to reduce flow completion times, or to maximize the utilization of a specific link), the constraints that need to be satisfied, and how to map OpenTCP updates to changes in individual TCP sessions. As part of congestion control policies, the operator can define a set of preconditions that need to be satisfied before any modifications are made to specific TCP sessions. This creates a controlled environment that the operator can use to modify TCP without the risk of making the network unstable.

Modify TCP at end-hosts: A key decision to be made here is how OpenTCP changes TCP's behaviour. The two options are (*i*) we can implicitly change TCP's behaviour by modifying the feedback provided to TCP sources (for example through ECN bits), or (*ii*) we can explicitly change TCP by having an *agent* running on each end-host that can update TCP on demand. The first option does not require any changes in the end-hosts, however, we don't have much flexibility in making the changes we want. As long as our end-hosts are under a single administrative control, changing them is feasible. This is true for example in a data center environment. As a result, we have designed OpenTCP under the assumption that we can change end-hosts and install a lightweight agent that can explicitly change TCP sessions. Taking advantage of today's extendible TCP implementations, we can easily modify TCP and even introduce completely new congestion control schemes when needed.

Two-timescale control: OpenTCP goes through a cycle of three steps: (*i*) data collection, (*ii*) CUE generation, and (*iii*) TCP adaptation. These steps are repeated with a pre-determined period of T seconds. The exact value of T is a choice of the network operator. In order to keep the network stable, T needs to be orders of magnitude slower than the network RTT. Intuitively, by slowly modifying TCP parameters, OpenTCP gives each TCP session enough time to become stable before updating its state. Moreover, by updating the system in a timescale which is orders of magnitude slower than the network RTT, OpenTCP can ensure a low overhead on the SDN controller.

Transitioning between TCP variants: OpenTCP can be used to choose between two or more variants of TCP based on network and traffic conditions. Let us consider K variants of TCP, say $P_1, P_2, \ldots, P_K$. Previous works by Tang *et al.* [26, 27] suggests that OpenTCP can keep the system stable as long as we have the following three conditions:

1. $\forall i, j : 1 \leq i, j \leq K$, P_i and P_j have *compatible congestion measures*;
2. $\forall i : 1 \leq i \leq K$, the network is stable under P_i; the utility functions are concave and monotonic; and
3. Changes are applied at a slow timescale (T) compared to the timescale of TCP window dynamics (RTT), which we call fast timescale.

Intuitively, having compatible congestion measures means that even though different TCP variants react to different congestion signals (*e.g.,* Reno to packet loss probability and FAST [29] to queueing delay) the measures can be related through a price mapping function as shown in [27]. Having a two timescale scheme, ensures that each flow can reach steady state before another round of changes is applied. We leave a more formal description of congestion measure compatibility and the proof of stability to future work.

OpenTCP and non-SDNs: OpenTCP is explicitly designed for SDNs. The existence of a centralized controller, with access to the global state of the network, and switches, which can be used to collect flow and link level statistics, make SDNs the perfect platform for OpenTCP. Having said that, one can use OpenTCP in a traditional network (non-SDN) as long as we can collect the required network state and traffic statistics. Clearly, this requires more effort and might have a higher overhead in a traditional network since we do not have a centralized controller and the built-in statistics collection features. As a workaround, in the absence of an SDN controller, we can use a dedicated node in the network to act as the Oracle. We can use SNMP to collect general link level statistics (*e.g.,* utilization and drop rate). Also, we can use CCAs to collect flow related information (*e.g.,* number of active flows). The Oracle can query the switches and CCAs to collect different statistics and if needed to aggregate them to match the statistics available in SDNs. This needs to be done with extra care, so that the overhead remains low, and the data collection system does not have a major impact on the underlying network traffic.

Feasibility, stability, and improvements: As a proof of concept, we have deployed OpenTCP in a HPC data center. We present a simple case study of OpenTCP that adaptively adjusts initial congestion

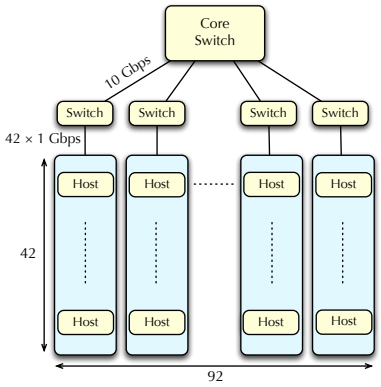

Figure 3: The topology of SciNet, the HPC data center network used for deploying and evaluating OpenTCP.

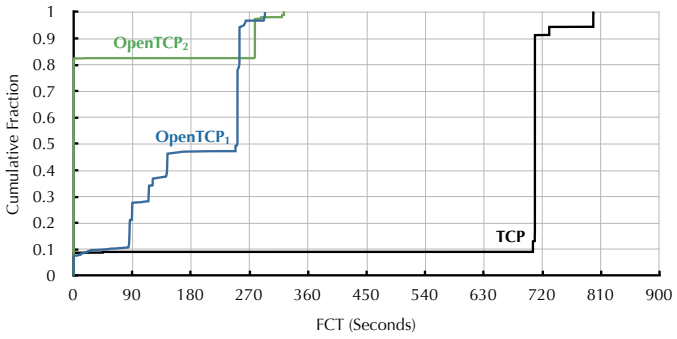

Figure 4: Comparing the CDF of FCT of $OpenTCP_1$, $OpenTCP_2$, and default TCP while running the "all-to-all" IMB benchmark.

window size and Retransmission Time Out (RTO). Throughout this study, we demonstrate how OpenTCP simplifies the process of adapting TCP to varying network conditions. Interestingly, such an adaption results in up to 59% improvement in flow completion times while keeping the network stable.

3. EVALUATION

In this section, we evaluate the performance of a preliminary OpenTCP implementation at SciNet [2] the largest HPC data center in Canada. Through a course of 20 days, we enabled OpenTCP and collected ∼200TB of packet level trace as well as flow and link level statistics.

Set-up: The topology of SciNet is illustrated in Figure 3. There are 92 racks, each of 42 servers. Each server connects to a Top of Rack switch (ToR) via 1Gbps Ethernet. The ToRs are Blade Network Technologies Gigabit switches. Each end-host in SciNet runs Linux 2.6.18 with BIC [30] as the default TCP. SciNet consists of $3,864$ nodes with a total of $30,912$ cores (Intel Xeon Nehalem) at 2.53GHz, with 16GB RAM per node. ToR switches are connected to a core Myricom Myri-10G 256-Port 10GigE Chassis.

Traffic Characterization: We recognize two types of co-existing TCP traffic in our traces: (i) MPI traffic and (ii) distributed file system flows. We measure that the majority of MPI flows are less than 10MB in size. In parallel to MPI flows, the distributed file system traffic ranges from 20B to 62GB in size. Moreover, we observe two utilization patterns in SciNet. First, 80% of the time, link utilizations are below 50%. Second, there are congestion epochs in the network where both edge and core links experience high levels of utilization, and thus packet losses. We find these patterns analogous to previous studies of data center traffic [3, 4, 24].

Methodology: We run a series of experiments to study how OpenTCP impacts flow completion time and packet drop rates. Throughout our experiments, we set OpenTCP's slow timescale (T) to 1min, unless otherwise stated, directing OpenTCP to collect statistics and send CUEs once every 1min. In this environment the fast timescale (RTT) is less than 1ms. To evaluate the impact of OpenTCP in an apples-with-apples fashion, we split the servers into two sets with OpenTCP enabled on one of these two sets only (*i.e.,* half of the nodes).

Benchmarks: In order to study different properties of OpenTCP, we use Intel MPI Benchmarks (IMB) [1] as well as sample user jobs in SciNet. These benchmarks cover a wide range of computing applications, have different processing and traffic load characteristics, and stress the SciNet network in multiple ways. We also use sample user jobs (selected from the poll of jobs submitted by real users) to have an understanding of the behaviour of the system under typical user generated workload patterns. Due to lack of space, the results presented here are for "all-to-all" IMB benchmark. We measured similar consistent results with other IMB benchmarks as well as sample user jobs.

Our first experiment with OpenTCP aims at reducing the Flow Completion Times (FCTs) by updating *init_cwnd* and RTO for TCP flows. We have observed that in SciNet, there is a strong correlation between MPI job completion times and the tail of FCT. Therefore, reducing FCTs will have a direct impact on job completion times. $OpenTCP_1$ collects link utilizations, drop rate, and number of live flows and sets the initial congestion window to 20 segments and RTO to 2ms as long as the maximum link utilization is kept below 70%. If the link utilization goes above 70% for any of the links, the Oracle will stop sending CUEs and CCAs will fall back to the default values of the initial congestion window size and the RTO.

While $OpenTCP_1$ aims at improving FCTs by adapting *init_cwnd* and ignoring packet drops, we define a second variant called $OpenTCP_2$ which improves upon $OpenTCP_1$ by keeping the packet drop rate below 0.1%. This means that when the CCA observes a packet drop

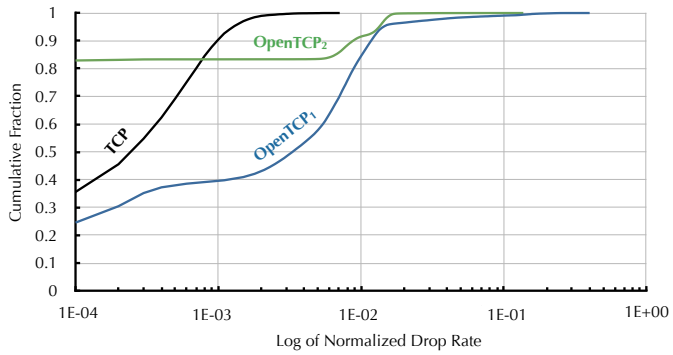

Figure 5: Comparing the CDF of drop rate of $OpenTCP_1$, $OpenTCP_2$, and default TCP while running the "all-to-all" IMB benchmark.

rate above 0.1% it reverts the initial congestion window size and the RTO to their default values.

Impact on flow completion time: Figure 4 depicts the CDF of flow completion times for $OpenTCP_1$, $OpenTCP_2$, and unmodified TCP. Here, the tail of FCT curve for $OpenTCP_1$ is almost 64% shorter compared to unmodified TCP. As we mentioned before, this leads to a faster job completion time in our experiments. Additionally, more than 45% of the flows finish in less than 260 seconds in $OpenTCP_1$, which indicates a 62% improvement compared to unmodified TCP. Similarly, $OpenTCP_2$ helps 80% of flows to finish in a fraction of a second which is a significant improvement, even compared to $OpenTCP_1$. This is because of $OpenTCP_2$'s fast reaction to drop rate at the end-hosts. In $OpenTCP_2$, the tail of the FCT curve is 324 seconds which is 59% improvement compared to the unmodified TCP. The FCT tail of $OpenTCP_2$ is slightly (8%) longer than $OpenTCP_1$ since it incorporates conditional CUEs and does not increase *init_cwnd* aggressively.

Impact on congestion window size: The improvements in FCTs are direct result of increasing *init_cwnd*. Our analysis showed that in the case of unmodified TCP, more than 70% of packets are sent while the source has a congestion window size of three, four, or five segments. OpenTCP however, is able to operate at a larger range for congestion window sizes: more than 50% of packets are sent while the congestion window size is greater than 5 segments (figure not shown due to space limitation).

Impact on packet drop rate: The FCT improvements in $OpenTCP_1$ and $OpenTCP_2$ come at a price. Clearly, operating at larger congestion window sizes will result in a higher probability of congestion in the network, and thus, may lead to packet drops. As Figure 5 shows, the drop rate distribution for unmodified TCP has a shorter tail compared to $OpenTCP_1$ and $OpenTCP_2$. As expected, $OpenTCP_1$ introduces a considerable amount of drops in the network since the CUEs are *not* conditional and thus the CCAs do not react to packet drops. $OpenTCP_2$ has no drops for more than 81% of the flows. This is a significant improvement over unmodified TCP. However, the highest 18% of drops in $OpenTCP_2$ are worse than those in unmodified TCP, as $OpenTCP_2$ needs to observe packet drops before it can react. Some flows will take the hit in this case, and might end up with relatively large drop rates.

Table 1: The approximate OpenTCP overhead for a network with ∼4000 nodes and ∼100 switches for different Oracle refresh cycles.

Overhead \ Oracle refresh cycle	1 min	5 min	10 min
Oracle CPU Overhead (%)	0.9	0.0	0.0
CCA CPU Overhead (%)	0.5	0.1	0.0
Data Collection (Kbps)	453	189	95
CUE Dissemination (Kbps)	530	252	75

Overheads of OpenTCP: We summarize OpenTCP's overhead for refresh periods of 1, 5, and 10 minutes in Table 1 by categorizing its overhead into 4 parts: (*i*) CPU overhead associated with data aggregation and CUE generation at the Oracle, (*ii*) CPU overhead associated with data collection and CUE enforcement at the end-hosts, (*iii*) required bandwidth to transfer statistics from the end-hosts to the Oracle, and (*iv*) required bandwidth to disseminate CUEs to the end-hosts. Clearly, OpenTCP's refresh cycle plays a critical role in its overhead on networking elements. We conclude that our implementation of OpenTCP has a small processing and bandwidth overhead.

4. RELATED WORK

Generally, congestion control in computer networks is handled in three different ways: (*i*) End-to-end congestion control protocols: these protocols use end-to-end network measurements, such as packet drops, as an implicit sign of congestion [18, 22, 23, 25]. (*ii*) Active Queue Management (AQM) schemes: here the routers along a flow's path have an active role in measuring congestion and signaling to the end systems [12, 14, 17, 20]. AQMs can provide better network utilization than drop-tail queue management if they are properly used, but can induce network instability and major traffic disruption, if not properly configured [11, 13, 20]. (*iii*) Hybrid of end-to-end and AQM schemes: these are techniques where routers along the path of a flow explicitly calculate the rate for individual flows, and notify the sources so that the sender can adjust its rate accordingly [3, 9, 10, 19].

OpenTCP is designed to use SDN as a platform. SDN proposals introduce several solutions to separate control and data plane in enterprise networks [5, 16, 21, 31]. However, much less effort has been put into congestion

control adaptation for this new environment; most proposals focus on routing and forwarding feature and ignore the advantages that SDNs provide in congestion control domain. OpenTCP bridges this gap and is designed to operate as a congestion control framework for SDNs by taking advantage of the luxuries provided in SDNs, such as programmable switches and centralized controller.

5. CONCLUSION AND FUTURE WORK

In this paper, we present OpenTCP, an automated TCP tuning framework for Software-Defined Networks. OpenTCP is flexible in terms of the changes it can apply. Our current implementation of OpenTCP is limited in scope, but can be extended to support programmable and adaptive TCP congestion control. To realize that, we plan to formalize general congestion control policies, and will provide abstract interfaces for network operators to program the behaviour of TCP. In that setting, network operators specify network policies and desired TCP behaviour, and OpenTCP adaptively and dynamically enforces them in the network.

Acknowledgements

We would like to thank Chris Loken, Daniel Gruner, Ching-Hsing Yu for help and support during our experiments at SciNet. We also would like to thank Phillipa Gill, Mukarram Bin Tariq and John Wilkes for comments and discussions.

6. REFERENCES

[1] Intel MPI Benchmarks, http://software.intel.com/en-us/articles/intel-mpi-benchmarks/.
[2] Scinet consortium, http://www.scinethpc.ca/.
[3] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). *SIGCOMM 2010*, pages 63–74.
[4] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. *IMC 2010*, pages 267–280.
[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, 2007.
[6] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. *WREN 2009*, pages 73–82.
[7] J. Chu. Tuning TCP parameters for the 21st century, http://www6.ietf.org/mail-archive/web/tcpm/current/msg04707.html.
[8] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Congestion Window, http://www.ietf.org/id/draft-hkchu-tcpm-initcwnd-01.txt.
[9] N. Dukkipati and N. Mckeown. RCP-AC: Congestion control to make flows complete quickly in any environment. *High-Speed Networking Workshop: The Terabits Challenge, IEEE INFOCOM 2006*, 2006.
[10] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, 2006.
[11] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring RED gateway. *INFOCOM 1999*, pages 1320–1328.
[12] W. Feng, K. Shin, D. Kandlur, and D. Saha. The BLUE active queue management algorithms. *IEEE/ACM Trans. Netw.*, 10:513–528, August 2002.
[13] S. Floyd. TCP and explicit congestion notification. *SIGCOMM Comput. Commun. Rev.*, 24:10–23, 1994.
[14] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1:397–413, August 1993.
[15] M. Ghobadi, G. Anderson, Y. Ganjali, J. Chu, L. Brakmo, and N. Dukkipati. TCP pacing in data center networks. Technical report, TR10-SN-UT-04-10-00, University of Toronto, April 2010.
[16] 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. *SIGCOMM Comput. Commun. Rev.*, 35(5):54, 2005.
[17] J. Hong, C. Joo, and S. Bahk. Active queue management algorithm considering queue and load states. *SIGCOMM Comput. Commun. Rev.*, 30:886–892, February 2007.
[18] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 25(1):157–187, 1995.
[19] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev.*, 32:89–102, August 2002.
[20] D. Lin and R. Morris. Dynamics of random early detection. *SIGCOMM Comput. Commun. Rev.*, 27:127–137, October 1997.
[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.
[22] J. Padhye, J. Kurose, D. Towsley, and D. Koodli. A model based TCP-friendly rate control protocol. *NOSSDAV99*.
[23] R. Rejaie, M. Handley, and D. Estrin. An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. *INFOCOMM 99*, pages 1337–1345.
[24] K. Srikanth, P. Jitendra, and B. Paramvir. Flyways to de-congest data center networks. *Hotnets'09*, 2009.
[25] W. Tan and A. Zakhor. Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE Transactions on Multimedia 1999*, 1(2):172–186.
[26] A. Tang, J. Wang, S. H. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: existence and uniqueness. *IEEE/ACM Trans. Netw.*, 15:824–837, August 2007.
[27] A. Tang, X. Wei, S. H. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: Optimality and stability. *IEEE/ACM Trans. Netw.*, 18:844–857, 2010.
[28] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. *SIGCOMM 2009*, pages 303–314.
[29] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, 2006.
[30] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *IEEE Infocom*, volume 4, pages 2514–2524, 2004.
[31] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: a 4D network control plane. *NSDI'07*, pages 27–27, 2007.
[32] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. *SIGCOMM Comput. Commun. Rev.*, pages 133–147, 1991.