RAPPEP: A Framework for Deploying Router-Assisted Congestion Control Protocols at TCP Performance Enhancement Proxy

Xiaolong Li Homayoun Yousefi'zadeh
Center for Pervasive Communications and Computing
University of California, Irvine
[xiaolonl, hyousefi]@uci.edu

***Abstract*— Router Assisted congestion control Protocols (RAPs) appear to be the most efficient solutions to the TCP performance degradation issue in high Bandwidth Delay Product (BDP) networks. Global deployment of RAPs such as XCP, VCP, and MPCP however, has been challenging due to their need for router support. In this paper, we propose RAPPEP a framework for deploying RAPs on potential congestion zones such as satellite links that are locally utilizing the architecture of TCP Performance Enhancement Proxy (PEP). Such a marriage allows for an immediate deployment of RAPs without the need for global router support, while still being able to take advantage of sophisticated RAPs. Beyond the deployed congestion zone, RAPPEP is completely transparent to the rest of the network including end nodes and other routers. Adapting from two implementations of RAPs and an implementation of TCP PEP (PEPSal), we implement and integrate RAPPEP in the Linux kernel and demonstrate its performance improvement compared to PEPSal through emulation studies.**

***Index Terms*— TCP PEP, XCP, VCP, MPCP, Router-Assisted Congestion Control Protocols, High BDP, Satellite Links.**

I. Introduction

Over the course of past few years, a wide variety of techniques have been proposed to combat the performance degradation of TCP in high BDP networks. The majority of these proposals remain as TCP variants that concentrate on either adaptively tuning the parameters of Additive-Increase Multiplicative-Decrease (AIMD) based on sophisticated control algorithms [1], [2], [3], or utilizing novel congestion detection mechanisms [4], [5], [6], [7]. Such techniques preserve the end-to-end nature of TCP and thus allow for an immediate deployment. However, since both the efficiency and the fairness are controlled by an integrated controller, they often fail to achieve high utilization and fairness while maintaining low persistent queue lengths and minimizing congestion-induced packet drop rates.

To that end, RAPs [8], [9], [10], [11] are proposed to decouple fairness control from efficiency. RAPs can achieve high utilization, low persistent queue length, insignificant packet loss rate, and sound fairness. Fundamentally, all RAPs require all intermediate routers to monitor network congestion status and derive appropriate feedback for the sender. Depending on the type of feedback information, RAPs can be classified into two general categories: i) protocols that utilize explicit feedback (typically sending rate) and ii) those that rely on load factors (typically the ratio of demand to capacity). We refer to the first category protocols as XFPs and the second category as LFPs. An example of the first category is eXplicit Congestion-control Protocol (XCP) while Variable-structure Congestion-control Protocol (VCP) [9] and Multi-Packet Congestion-control Protocol (MPCP) [10] represent examples of the second category.

Since feedback needs to be encapsulated in the header of acknowledgment packets, all RAPs require extra bits in the packet header. Such requirement in conjunction with the need to receive support from all intermediate routers introduces substantial deployment obstacles for RAPs.

Deviating from these pure end-to-end solutions introduced above, TCP PEP schemes [12], [13] have emerged to improve the end-to-end performance of TCP, where large delay, moderate bandwidth links such as satellite links are in use. By splitting an end-to-end TCP connection to multiple segments, TCP PEP is able to isolate the satellite portion from the rest of the network. Such splitting allows for using specially designed congestion control algorithms suitable for satellite links and thus improving end-to-end performance.

Because of being non-intrusive and immediately deployable, TCP PEP has been widely deployed in today's satellite networks and is currently considered to be among industry's most current practices. However and as discussed in Section IV, TCP PEP is inefficient in comparison to RAPs due to the integrated nature of TCP congestion controller. In essence, there is a two fold rationale for the use of TCP PEP: i) isolation of TCP-unfriendly zone where TCP variants fail to work well, and ii) utilization of specially designed congestion control algorithms in the TCP-unfriendly zone. Intuitively, TCP PEP provides an ideal environment for deploying RAPs locally, which not only facilitates the deployment of RAPs but also further improves the performance of TCP PEP by utilizing sophisticated RAP control algorithms.

In this paper, we propose and implement RAPPEP, a frame work that deploys RAP algorithms transparently in TCP PEP. The transparency comes from the fact that RAPPEP requires changes to neither existing TCP PEP architecture, nor the introduction of new TCP options and/or transport protocols in the network stack. Simply put, all RAPs operate exactly like a standard TCP variant in RAPPEP. Our paper makes several key contributions with regards to both RAPs and TCP PEP. First, we propose a framework that adapts a RAP-based implementation within TCP PEP architecture. It removes the need for changing standard TCP options and/or introducing new transport protocols as [14], [15]. To the best of our knowledge, this is the first work on applying RAPs to TCP PEP. Second, this paper reports on the implementation of an XFP and an LFP algorithm fol-

This work was sponsored in part by a research grant from the Boeing Company.

lowing the proposed framework. While the XFP algorithm utilizes XCP, the LFP algorithm adapts MPCP. Most importantly, all RAP algorithms are implemented as standard TCP flavors in Linux utilizing the Linux TCP pluggable congestion control structure. This allows for the co-existence of RAPs and other transport protocols. Third, the paper conducts extensive experimental studies comparing two RAPPEP implementations with raw RAPs as well as raw TCP PEP. It is our belief that an approach such as RAPPEP can improve the appeal of RAPs by offering a transparent and tangible benefit that can significantly improve efficiency and fairness of TCP congestion control over high BDP links.

The rest of the paper is organized as follows. Section II presents the fundamentals of RAPs and TCP PEP. Section III presents the implementation approach of the RAPPEP. Experimental studies are presented in Section IV. Finally, we present several conclusions in Section V.

II. Background of RAPs and TCP PEP

In this section, we briefly review the key concepts of RAPs and TCP PEP. The implementation approach of XCP, VCP, and MPCP are introduced as well.

A. LFPs: VCP and MPCP

Recently, a family of LFPs including but not limited to VCP, MPCP, and MLCP have emerged. Fundamentally, all LFPs need to calculate the Load Factor (LF) of a link which is represented as quantized feedback in a number of bits. These protocols mainly differ in two aspects: the number of quantized feedback bits; and the way the feedback is generated and delivered. In this subsection, we briefly describe VCP and MPCP.

VCP regulates the $cwnd$ with different congestion control policies according to the level of congestion in the network. Three levels of congestion low-load, high-load, and overload are mapped into two ECN bits of the IP packet header. A VCP-capable router computes the LF of each of its links and maps each LF to one of the three congestion levels. Upon arrival of a data packet from a sender, each router examines the congestion level of its upstream link carried in the ECN bits of the packet and updates ECN bits only if its downstream link is more congested. Finally, the receiver signals the sender with the congestion information via acknowledgment (ACK) packets. Consequently, VCP applies three congestion control policies: Multiplicative-Increase (MI) in the low-load region, AI in the high-load region, and MD in the overload region. The MI region is utilized to eliminate the slow start characteristic of TCP while AI and MD regions preserve the fairness characteristics of TCP. Since VCP can only provide limited feedback to the sender, its efficiency and fairness characteristics are negatively impacted in moderate bandwidth high delay network operation scenarios.

Multi Packet Congestion control Protocol (MPCP) is developed to overcome the limitations of VCP. In MPCP, the transmitting side can receive finer-grained congestion information without requiring the use of any additional bits in the IP header beyond the two ECN bits. Specifically, a congestion level is carried by a chain of n packets and each packet provides two bits out of $2n$ bits of information associated with a congestion level. While routers compute and distribute congestion signaling into n packets, end nodes retrieve a congestion level by concatenating a group of $2n$ ECN bits together from a set of packets. MPCP correctly handles the packet ordering and reacts to network exceptions such as packet loss and out of ordering delivering. MPCP is already implemented for $n \in \{2, 3\}$.

As shown in Section IV, VCP does not perform well in satellite environment and as such we only consider adapting MPCP algorithm for RAPPEP.

B. XFPs: XCP

Unlike LFPs, XFPs explicitly provide the sender with the next sending rate. Thus, the task of sending rate control is offloaded from the end nodes to the routers. Since all of the XFPs provided in the literature are essentially variants of XCP, we focus on XCP and its implementation approach in this subsection. Initially, an XCP sender encapsulates its current $cwnd$, the sender perceived RTT, and the bandwidth demands in the packet header. The bandwidth demands of the sender can be modified by routers according to the congestion status of the network and finally contain the feedback. As a result, XCP requires the use of multiple bits in the IP header of each packet. An XCP router has two separate controllers: the Efficiency Controller (EC) and the Fairness Controller (FC). First, the EC computes the aggregate feedback according to the input traffic rate, the persistent queue length, and the link capacity. Then, the FC allocates the aggregate feedback to individual packets in order to achieve fairness. Notably, XCP uses Multiplicative-Increase Additive-Increase Multiplicative-Decrease (MIMD) for efficiency and AIMD for fairness control. In [14], XCP is implemented in the Linux kernel with 16 bits of feedback. XCP is implemented with two components: XCP router and XCP end host. While the XCP router is implemented as two LKMs utilizing Linux $Qdisc$ architecture a built-in Linux packet queuing and scheduling mechanism, the functionality of XCP end host is implemented by introducing a new TCP option.

C. TCP PEP

TCP PEP schemes utilize network agents installed on intermediate nodes of a network path to improve the end-to-end performance of TCP. Among the PEP solutions, the splitting approach [13] appears to be the most promising, where a TCP connection is split into multiple segments. While the distributed PEP utilizes two PEP agents and splits a connection to three segments, the integrated PEP only deploys one PEP agent and splits a connection into two segments. Since we implement RAPPEP on top of the PEPsal, an integrated PEP implementation, we focus on the integrated PEP in this subsection. Typically, a PEP agent sits at the uplink gateway of a satellite link. The first TCP segment takes advantage of the standard TCP and is terminated at the PEP agent. The second segment originates from the PEP agent and is terminated at the end receiver. Notably, if the same techniques of congestion control are applied to all three segments, the end-to-end throughput will not be improved significantly [13]. Only the performance of the

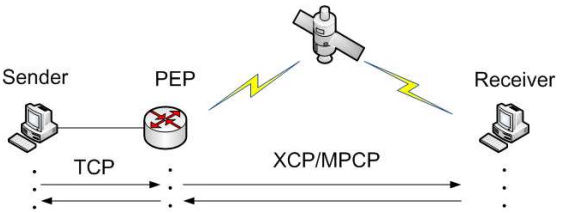

Fig. 1. An illustration of the RAPPEP framework.

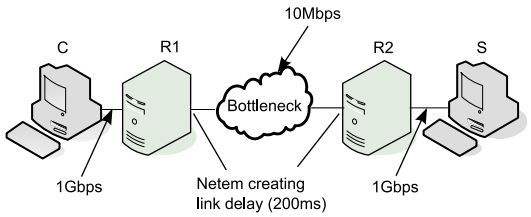

Fig. 2. An illustration of the experimental environment.

first segment can be improved due to the fact that the sender's congestion window can be opened much faster because of the early ACKs from the proxy. Thus, the PEPsal proposed to use TCP Hybla [16] with local retransmission [17], selective ACK [18] *only* on the segment with the highest delay. As noted earlier, the use of splitting approach allows for using specially designed congestion control algorithms that are suitable for satellite links.

III. RAPPEP

RAPPEP is a framework that can facilitate the integration of RAP control algorithms with the PEP architecture. In this paper, we build RAPPEP on top of the PEPSal, which is an open-source implementation of an integrated PEP. As noted earlier, a PEP agent has to operate as the terminator and the initiator of two TCP segments simultaneously while RAP implementations distribute the control functionality to both routers and end nodes. In order to apply RAP algorithms to PEP, RAPPEP has to deal with following challenges:

1) The co-existence of both standard TCP congestion control algorithms and new RAP algorithms. A TCP PEP agent has to be able to adjust $cwnd$ using RAP algorithms for a flow initiated from the agent, while processing data packets and replying with ACK packets using the standard TCP algorithm for a flow terminated at the agent.
2) The distribution of RAP functionality. While applying RAP algorithms, the TCP PEP agent has to function as both end nodes and routers simultaneously.
3) The implementation of a RAP algorithm on top of PEPsal.

A. RAPPEP Overview

Fig. 1 illustrates the architecture of the RAPPEP. Note that we implement both XCP and MPCP algorithms in RAPPEP, to which we refer as XCP PEP and MPCP PEP, respectively.

Intuitively, it might appear that one can directly port the existing XCP/MPCP implementations to a PEP agent. As shown in the Fig. 1, XCP/MPCP needs to be deployed at the Receiver as well. Thus, such approach would fail to work since the PEP agent, one of the end nodes of the XCP/MPCP connection, would not be able to receive any feedback from the Receiver. Consequently, the agent cannot adjust the $cwnd$ due to the absence of intermediate routers that compute and relay feedback. However, such failure provides an important premise for a transparent implementation of RAPPEP because the Receiver can be potentially freed from processing and relaying the feedback to the sender. Specifically, in an end-to-end scenario, the PEP agent is actually an XCP/MPCP router that monitors the congestion status of the satellite link and computes the feedback. Notably, as the initiator of the second XCP/MPCP segment in the PEP scenario, the PEP agent would function as an XCP/MPCP sender. If the PEP agent can function as an XCP/MPCP router, the requirement of forwarding the feedback can be eliminated. More specifically, the agent perceived feedback coming from the receiver is the same as what was computed earlier at the agent since the agent is the *only* router of the connection. Thus, there is no need to encapsulate the next sending rate/load factor in the packet sending to the receiver and extract it from the feedback. Instead, functioning as both a sender and a router, the PEP agent can obtain the expected feedback *locally* rather than retrieving the feedback from an ACK. As a result, the receiver is completely freed from handling the feedback and can use any standard TCP variant allowing for compatibility. Meanwhile, the limitation of the availability of extra bits in the IP packet header can be removed as well benefiting both XFP and LFP. While LFPs is allowed to use the accurate LF for the purpose of control, XFP can use any number of bits to store the feedback. Consequently, RAPPEP is implemented as a new standard TCP congestion control algorithm utilizing the Linux Pluggable congestion control architecture.

B. RAPPEP Implementation

We implement XCP and MPCP algorithms in RAPPEP as LKMs. Specifically, RAPPEP consists of two components: RAPPEP Traffic Monitor (RTM) and RAPPEP Controller (RC). The RTM utilizes $netfilter$ to monitor the traffic rate and the persistent queue length. In contrast to the raw MPCP and XCP implementations, RTM monitors the outgoing traffic rate instead of input traffic rate. Note that RTM can serve for both XCP PEP and MPCP PEP. While RTM outputs the spare bandwidth for XCP PEP, the raw LF is output for MPCP PEP. The output of RTM is stored locally in a shared buffer at the PEP agent and will not be encapsulated in any packet.

The RC component computes the next sending rate based on the output of the RAPPEP traffic monitor and regulates the $cwnd$ of the agent. While the agent no longer needs to retrieve the feedback encapsulated in ACK packets, it still relies on the ACK packets to keep the algorithm semantics and the control timing. More specifically, upon arrival of an ACK packet, the RC component initiates an adjustment to $cwnd$ and directly uses saved RTM output instead of the information in the ACK. We note that there will be no congestion-caused loss for a one-hop TCP connection. Hence, RAPPEP treats all losses as wireless link effect loss and makes no adjustment to $cwnd$.

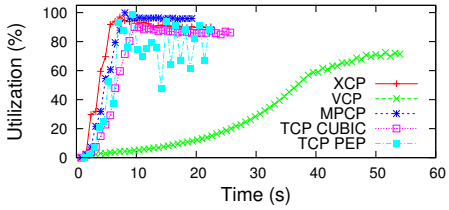

Fig. 3. A bottleneck utilization comparison of all protocols without loss.

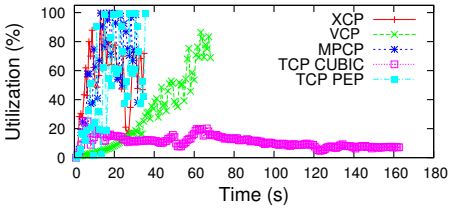

Fig. 4. A bottleneck utilization comparison of all protocols with 1% loss.

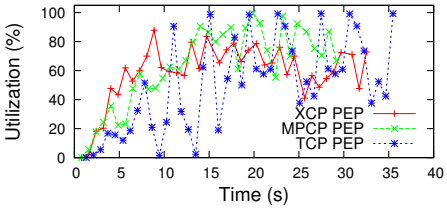

Fig. 5. A bottleneck utilization comparison of three PEP solutions with 1% loss.

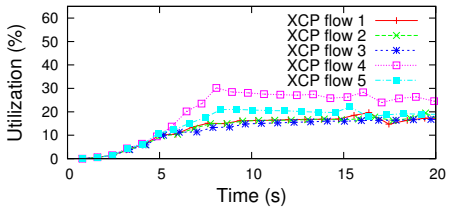

Fig. 6. Per-flow bottleneck utilization of XCP.

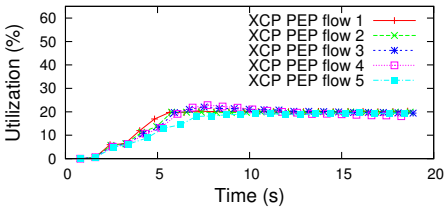

Fig. 7. Per-flow bottleneck utilization of XCP PEP.

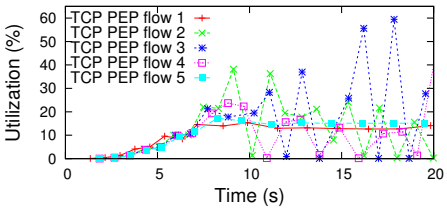

Fig. 8. Per-flow bottleneck utilization of TCP PEP.

Furthermore, since all flows over the second PEP connection traverse the same satellite link, they experience nearly the same link delay. Thus, for the control algorithms of both XCP and MPCP, there is no need to scale the control parameters for heterogeneous RTT scenarios, which significantly simplifies the implementation. Most importantly and as shown in Section IV, such characteristic can improve the fairness of RAPPEP by shielding RTT variations among flows.

Finally, we would like to summarize how the RAPPEP is designed and implemented to addresses the three challenges presented at the beginning of this section. First, by taking advantage of the Linux pluggable congestion control architecture, the RAPPEP Controller is integrated into the existing Linux congestion control algorithms, and thus allowing for the coexistence of both RAP and TCP. Secondly, in RAPPEP, instead of relaying the feedback information, RAP agent obtains the feedback locally and thus removing the need to implement RAP end nodes and routers separately. Finally, the RAPPEP is implemented in same way as other other standard TCP variants, i.e. utilizing the Linux pluggable congestion control architecture, which allows for seamlessly integration with the Pepsal.

IV. Experimental Studies

In this section, we present our experimental study conducted in a Linux testbed emulating satellite link effects. First the performance of VCP, XCP, MPCP, TCP CUBIC and TCP PEP are compared verifying the motivation of this paper. Then, utilizing the implementations of two RAPPEP schemes, namely, XCP PEP and MPCP PEP, we show how RAPPEP can further improve the performance of TCP PEP. Fig. 2 shows our single bottleneck experimental setup.

All of the nodes utilize Fedora Core 5 (FC5) distribution of Linux. For TCP PEP experiments, we use the open-source implementation of PEPsal [13]. A packet loss rate of 1% is introduced utilizing the NistNet emulator [19] as presented in [13]. The 10 Mbps bandwidth of the full-duplex bottleneck link between R1 and R2 is controlled by the $ethtool$ interface available in FC5. Bottleneck link delays are introduced utilizing $netem$ utility [20] on both routers. The one way link delay is set to 400 ms. Both side links from an end host to a router operate in full-duplex mode and have a bandwidth of 1 Gbps. Five FTP flows start from one end host to another end host at random times with each flow carrying a 5 MB file. FTP servers are set up at both end hosts using $vsftpd$ daemon available in FC5. For XCP involved experiments, a pair of customized XCP-capable ftp client and $vsftpd$ are used.

To optimize the performance of TCP, a variety of TCP parameters are fine tuned following TCP performance tuning guides of [21] and [22]. While XCP, VCP, and MPCP parameters are set as presented in [14], [15], [10], TCP PEP parameters are set as presented in [13]. In the case of PEPs, PEPsal runs at R1 and R2 with TCP Hybla, XCP PEP, and MPCP PEP algorithms, respectively. In the experiments, we use bottleneck link utilization, Average FTP Completion Time (AFCT), and congestion window size as performance metrics. TCP CUBIC is used at end nodes for PEP experiments as it performs best among TCP variants.

Fig. 3 shows the bottleneck link utilization for each protocol without introducing packet loss. All RAPs outperform TCP CUBIC and TCP PEP except VCP due to its slow speed of convergence associated with the high delay. The end point of each curve indicates the moment when the last FTP flow completes, but not the AFCT of each protocol since not all flows complete at the same time. In this experiment, the AFCT of MPCP is 18 seconds 25% faster than that of TCP PEP. This performance gap between RAPs and TCP PEP verifies the motivation of this paper presented earlier. Fig. 4 illustrates that similar patterns are observed when 1% packet loss is introduced. While all protocols show oscillatory behavior due to the introduction of loss, TCP PEP demonstrates the worst oscillatory behavior because it is unable to distinguish between congestion caused loss and link quality caused loss. More importantly, although it takes all protocols a longer time to complete file transfer, their performance gap increases as well. The AFCT of both MPCP and XCP are around 24 seconds 35% faster than that of TCP PEP.

Fig. 5 compares the bottleneck link utilization of each PEP

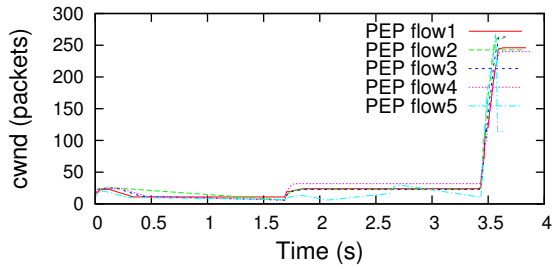

Fig. 9. An illustration of cwnd dynamics at the FTP server.

solution. In contrast to 4, all of RAPPEP solutions offer good performance especially in terms of their oscillatory behavior. While eventually they achieve comparable utilization as their original alternatives, they converge faster because of the improvements presented in Section III. In the case of MPCP PEP, fewer oscillations are observed due to the absence of order handling and the effects of packet loss on LF. In the case of XCP PEP, a faster completion and fewer oscillations are observed because of simplifying bandwidth allocation, using a fixed RTT estimate, and avoiding the involvement of end nodes. We would like to note that while XCPPEP takes longer time to finish than MPCPPEP, it achieves a better fairness due to the use of a separate controller.

To better evaluate the fairness of each PEP schemes, we decrease the loss rate to 0.1% while varying the delay of each flow by introducing random delays on the side links range in the range of $[0, 800]$ ms. Notably, the link delay of the satellite link remains same. Fig. 6, 7, and 8 compare per flow bottleneck utilization of XCP, XCP PEP, and TCP PEP schemes. Notably, XCP PEP significantly improves the fairness of flows in contrast to TCP PEP due to use of a separate controller for fairness. In contrast to XCP, the use of the PEP hides the impact of the RTT difference on the first segment and achieves a significantly better fairness on the bottleneck link. For the same reason and while not shown here, MPCP PEP shows similar fairness improvement.

Finally, it is interesting to show the $cwnd$ dynamics at the sender while the PEP schemes are applied. As shown in Fig. 9, the $cwnd$ of each flow opens very fast and the transmission finishes in 5 seconds because PEPSal sends ACKs before the receiver gets the packet. Such behavior shields RAP PEPs from RTT variations.

It is important to note that the choice of RAP algorithm for RAPPEP represents a tradeoff. In our experiments, MPCP PEP demonstrates an efficiency better than that of XCP PEP showing a shorter AFCT because of its aggressive MI policy, while XCP PEP outperforms MPCP PEP in terms of fairness because XCP utilizes a separate controller for fairness control.

V. Conclusion

In this paper, we proposed RAPPEP, a framework of deploying Router-Assisted congestion control Protocol (RAP) at TCP Performance Enhancing Proxies. We showed that RAPPEP could circumvent the deployment limitations of RAPs by integrating RAP congestion control algorithms within TCP PEP agents while significantly improving the performance of TCP PEP measured in terms of bandwidth utilization and fairness. We implemented RAPPEP with an explicit feedback based algorithm (XCP) and a load factor based algorithm (MPCP) using an open source TCP PEP implementation (PEPsal) available in the Linux kernel. Through experimental studies, we demonstrated that RAPPEP represents a highly efficient solution that allows for immediate deployment of newly developed RAPs. Furthermore, we showed that RAPPEP could achieve good fairness by filtering the impact of flow RTT variation at the second TCP segment. Finally, it is our belief that a framework such as RAPPEP can improve the appeal of RAPs by offering a transparent and tangible benefit that can significantly improve TCP performance over high BDP networks.

References

[1] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *Proc. of the IEEE INFOCOM*, 2004.

[2] I. Rhee and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," in *Proc. of the PFLDNet'05*, Feb. 2005.

[3] S. Floyd, "HighSpeed TCP for Large Congestion Windows," Aug. 2002.

[4] D. Leith and R. Shorten, "H-TCP: TCP for High-speed and Long-distance Networks," in *Proc. of the PFLDNet'04*, Feb. 2004.

[5] T. Kelly, "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks," Feb. 2003, available at http://wwwlce.eng.cam.ac.uk/ctk21/scalable/.

[6] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proc. of IEEE INFOCOM*, 2004.

[7] S. Bhandarkar, S. Jain, and A. Reddy, "Improving TCP Performance in High Bandwidth High RTT Links Using Layered Congestion Control," in *Proc. of the PFLDNet'05*, Feb. 2005.

[8] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. ACM SIGCOMM*, Aug. 2002.

[9] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One More Bit Is Enough," in *Proc. ACM SIGCOMM, 2005*, Aug. 2005.

[10] X. Li and H. Yousefi'zadeh, "MPCP: Multi Packet Congestion-control Protocol," *ACM Computer Communications Review (CCR)*, vol. 39, no. 5, pp. 6–11, Oct. 2009.

[11] I. A. Qazi and T. Znati, "On the design of load factor based congestion control protocols for next-generation networks," in *Proc. of the IEEE INFOCOM 2008*, Apr. 2008.

[12] A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," in *Proc. of 15th International Conference on Distributed Computing Systems(ICDCS)*, Vancouver, BC, May 1995, pp. 136 – 143.

[13] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: A Performance Enhancing Proxy for TCP Satellite Connections," in *IEEE A and E SYSTEMS MAGAZINE*, Aug. 2007.

[14] Y. Zhang and T. Henderson, "An Implementation and Experimental Study of the eXplicit Control Protocol (XCP)," in *Proc. IEEE INFOCOM, 2005*, Mar. 2005.

[15] X. Li and H. Yousefi'zadeh, "An Implementation and Experimental Study of the Variable-Structure Congestion Control Protocol (VCP)," in *Proc. of the IEEE MILCOM, 2007*, Oct. 2007.

[16] C. Caini and R. Firrincieli, "TCP Hybla: a TCP Enhancement for Heterogeneous Networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, Sept. 2004.

[17] S. Keshav and S. P. Morgan, "SMART Retransmission: Performance with Overload and Random Losses," in *Proc. INFOCOM 1997*, Apr. 1997.

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "Selective acknowledgment options," in *RFC-2018*, Oct. 1996.

[19] M. Carson and D. Santay, "NIST Net-A Linux-based Network Emulation Tool," *Computer Communication Review*, June 2003.

[20] S. Hemminger, "Network Emulation with NetEm," in *Proc. LCA, 2005*, Apr. 2005.

[21] J. Mahdavi, "Enabling high performance data transfers on hosts," *technical note, Pittsburgh Supercomputing Center*, Dec. 1997, available at http://www.psc.edu/networking/perf_tune.html.

[22] -, "TCP Tuning Guide," Nov. 2005, distributed Systems Department, Available at http://dsd.lbl.gov/TCP-tuning/TCP-tuning.html.