To appear in IEEE GLOBECOM 2001. Copyright 2001 IEEE

An Active Proxy Based Architecture for TCP in Heterogeneous Variable Bandwidth Networks

Debojyoti Dutta   Yongguang Zhang
HRL Laboratories, LLC., Malibu, CA 90265, USA
ddutta@isi.edu   ygz@hrl.com

Abstract— **In heterogeneous networks such as today's Internet, TCP must be made to handle links and paths with vastly different characteristics, including longer delays and dynamically changing available bandwidth. In this research, we first analyze the performance of TCP under networks with TCP-unfriendly characteristics and introduce an architecture to tackle the performance problems in such networks.**

I. Introduction

Some networks cannot be easily incorporated into the present TCP dominated Internet infrastructure transparently since they possess new link technologies that have characteristics which can seriously impair the performance of TCP flows, e.g. networks with long delay links and/or with highly variable bandwidth. TCP performs very well in networks with fixed bandwidths and small delays. However, the bandwidth available to a TCP flow in today's Internet is often variable. For example, in a QoS network where TCP flows are multiplex-ed in the *best effort* category, the available bandwidth fluctuates when higher priority flows and other TCP flows come and go; in wireless networks where frequent mobility leading to vertical hand-offs, both bandwidth and delay change from an end-to-end [1] perspective. We call such networks with high variations in bandwidth and large delays as being *TCP-unfriendly*.

In this research we study the behavior of TCP under the above *unfriendly* network conditions and develop an architectural solution to improve its performance through router assisted dynamic congestion control. In particular, we use *Performance Enhancement Proxies (PEP)*.

II. Problem Description

In this section, we first explore the effects of *unfriendly* network conditions on TCP flows with NS2 simulations [2]. Our test network is illustrated in Figure 1. The links between A and C and between D and B simulate the local access networks and they are fixed at 10Mbps with 10ms latency. The link between C and D simulates the path through the heterogeneous networks in the Internet, and it has longer delay (parameter d) and dynamically variable bandwidth. We also make this link bottleneck part of the network so that its characteristics controls the the end-to-end performance between A and B.

We implement the variable bandwidth link with a new link object that we introduced in NS2. It calculates the instantaneous bandwidth whenever a packet is transmitted through it.

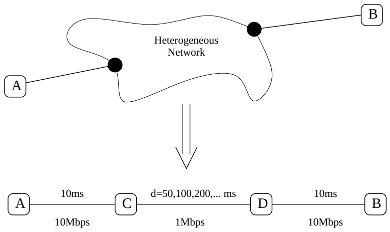


Fig. 1. The Simulation Network for Our Experimental Study

The calculation is based on some mathematical functions (such as $sine$) to model bandwidth oscillations. While a sine function may not be the most suitable approximation for studying the bandwidth dynamics in the Internet, its versatility allows us to see quickly how TCP responds to slow or fast changing bandwidth. For example, the oscillations that we used in our experiments (see Figure 2) are generated by fixing the mean at 1Mbps, the amplitude at 0.5Mbps, the phase at zero, and varying only the angular frequency (k) from 0.0 to 1.0 (second).

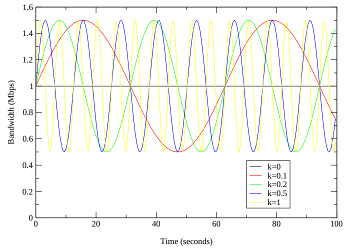


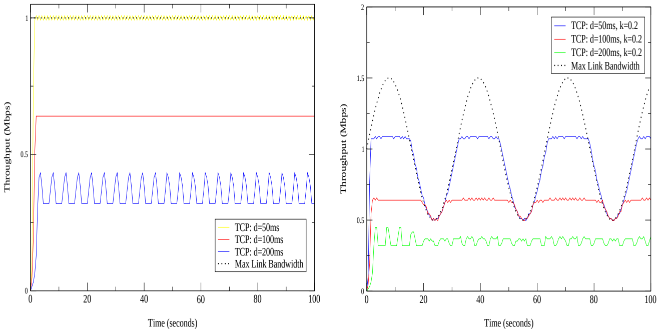
Fig. 2. Modeling Bandwidth Oscillation with Sine in NS2

We consider one long TCP flow from A to B. We use standard TCP configuration, i.e., TCP NewReno with window size 20, the initial congestion window 1, and Droptail queue size 50 for every link. We compare the effective throughput as seen by the receiver (at B).

A. TCP Suffers with Long Delays and Variable Bandwidth

We first look at the behavior of TCP flows under different delays but with no bandwidth oscillation. From Fig. 3(a) we clearly see that as the delay becomes longer the throughput becomes lower. This is because once the delays become high, we will need a larger TCP window, and a larger buffer at the bottleneck link to reach the maximum throughput level allowed by the available bandwidth. In fact TCP modeling tells us that

Debojyoti is also a student at USC/ISI.



(a) no bandwidth oscillation

(b) with bandwidth oscillation

Fig. 3. Effect of Link Delay d on TCP Performance

throughput T as function of the window size W and the Round Trip Time RTT is given by $T = W/RTT$

The rate of *slow start* [3] also increases with end to end delays. As the delay becomes longer, the slow start period lasts longer, resulting in sub-optimal performance at the beginning. This is very significant because the slow start period plays a major role in short TCP sessions, especially in web traffic. Thus large delays affect TCP performance.

In the second set of simulations, the purpose is to see how TCP responds to the same bandwidth oscillation when the network delay becomes larger. Fig. 3(b) illustrates the results on TCP throughput. This graph shows that, as the delay increases, the TCP sessions become less and less responsive to the bandwidth oscillation. All the space between the maximum link bandwidth curve and the throughput measurement curve represents the wasted bandwidth.

These simulations validate our prognostication - TCP does not respond effectively to very long delays, and, more importantly, variations in available bandwidth. The intrinsic reason for this behavior lies in the TCP's end-to-end congestion control model [3]. *Standard TCP* uses the timing of ACK packets to estimate the current available bandwidth in the forward path, and responds by applying linear increase and multiplicative decrease to the sender window size. Given the small buffer size of *standard TCP*, this probe-and-estimate method can become largely ineffective because the response time can be slower than the rate of change of bandwidth. The buffers may not be able to handle this delay and/or the bandwidth estimation may be stale. The result is that the TCP performance suffers.

III. New Architectural Solution

Our approach for improving TCP performance is to deploy TCP Performance Enhance Proxies (*PEP* [4]) at the edges of TCP-unfriendly networks (see Fig. 4) to control the TCP flows passing through them. The basic idea is for the PEP element at an edge router to monitor the available bandwidth of the network, and to manipulate the ACK packets of a passing-thru TCP flow accordingly. By deliberately speeding up or slowing down TCP ACKs, the PEP element can rewrite the behavior of a TCP flow and help achieve better congestion control and improved throughput.

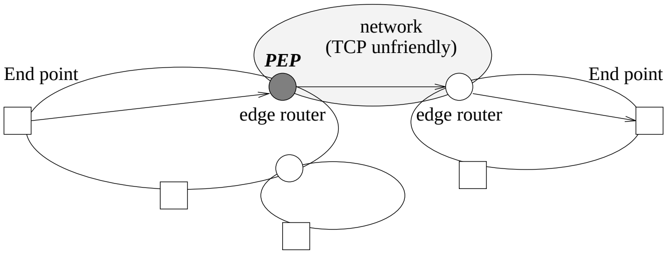


Fig. 4. A PEP-Based Architecture for Improving TCP Performance

The philosophy behind this approach is as follows. As we have shown earlier, TCP's open-loop congestion control becomes ineffective when the network delay becomes significantly large and when the available bandwidth becomes volatile. Therefore, if a network can provide more accurate hints on the current congestion condition, the TCP senders at the end points can respond more effectively.

First, such hints can only come from ACK manipulations inside the network because we are constrained from changing TCP end points (also see discussion in Section V). Whenever the network changes the timing of ACK arrival at the sender, it changes the subsequent behavior of the TCP flow.

Second, these manipulations should only come from the TCP-unfriendly networks because a network should always know its own condition better than any other node or end point. Further, limiting changes to only those networks that need such changes has an obvious advantage of being scalable.

Finally, the ingress router at the edge of a network/subnet is the best place for ACK manipulations for that network/subnet, because the control loop (sender to PEP to sender) can be shortened before it pass through the TCP-unfriendly links. Moreover, the PEP element can couple with other traffic control elements at the ingress router, such as admission control and scheduling, to provide a better congestion control.

A. Congestion Control in the PEP Element

In the big picture, the PEP element at the network edge monitors all TCP flows going in and out of the network (Fig. 4) and manipulates the TCP ACK packets for performance gains. Our design assumes that congestion usually happens at the edges. Hence we do not run into scalability issues.

A basic technique for TCP ACK manipulations is the *premature ACK* – a TCP acknowledgment packet manufactured by the PEP element but identical to the actual ACK generated by the TCP receiver. Usually, a TCP data segment must reach the destination to trigger the receiver to send a TCP ACK packet back to the sender. By contrast, a premature ACK is usually manufactured when the corresponding data segment passes by the PEP element; that means, while the data segment is still in transit to the receiver, the sender may already have received the ACK. This way, the premature ACK technique generates an effect that makes the network delay significantly lower, allowing the TCP congestion control to respond faster.

When the premature ACK technique is used, the burden falls upon PEP to recover any data that might be lost after the PEP element acknowledges it. Therefore, the PEP element must keep a buffer for all such segments that have been prematurely ACKed but for which the corresponding real ACKs have not arrived from the receivers. When a real ACK arrives, the PEP element must clear the corresponding data segment from its buffer and drop this real ACK (because the sender should have gotten the premature ACK earlier). It must also watch for other conditions that may indicate data losses, such as timeouts and three-duplicated-ACKs. Furthermore, the PEP element must control the rate of premature ACKs so that the TCP send rate will not be overly aggressive as to overrun the PEP buffer.

For every TCP flow passing through the PEP element, it uses a flow classifier to select the flows that need PEP enhancement. The selection policy can be defined using out-of-bound mechanisms and it is outside the scope of this study. For each TCP flow that needs PEP enhancement, the PEP element maintains a state for the flow. The state consists of a shared buffer to store TCP segments and a watermark index. This state information will be used in the ACK manipulation for the TCP flow.

When a TCP flow first starts, the PEP element queries other traffic control elements (such as the admission control) to find out the maximum bandwidth available to this flow. If the network is an integrated service network, such information may be available at the RSVP table [5]. Or, if the network is a best-effort-only network, the maximum bandwidth will be the total capacity of the bottleneck link. Once this number is obtained, the PEP element then accordingly set the maximum size for the shared buffer and an initial value for the watermark index.

When a TCP data segment arrives at the edge router, the PEP element will do one of the following, depending on the current buffer length and the watermark index:

- If the current buffer length is less than the watermark index, PEP makes a copy of this segment to store in the buffer, and generates a premature ACK for the sender. PEP marks this segment as “p-acked.”
- If the current buffer length is greater than or equal to the watermark index, but less than the the maximum buffer size, PEP also makes a copy to store in the buffer, but it does not generate a premature ACK. It marks this segment as “stored.”
- If the current buffer length is greater than or equal to the maximum buffer size (i.e., the buffer is full), PEP does nothing (i.e., let the data segments pass as if there were no PEP).

When a real TCP ACK packet arrives at the PEP element, the corresponding data segment (and any segments with a lower sequence number unless the session uses selective acknowledgment) will be fetched and purged from the buffer. If this data segment is already “p-acked”, the router will drop this ACK packet. In addition, since this can reduce the buffer length, some data segments that are previously marked “stored” may now fall below the watermark. The PEP element will then generate a new premature ACK for these segments and change their states from “stored” to “p-acked.” That is, the PEP element sends “hints” to the sender to send more data.

B. *The Watermark*

From the above discussion we can see that the watermark index is an important variable in the PEP congestion control. The watermark is a central point of our design. The watermark index divides the PEP buffer into two parts. When the buffer length falls below the watermark, it indicates that the TCP send rate is over pessimistic. Therefore, the PEP element always generates premature ACKs in this case. When the buffer length rises above the watermark, it indicates that the TCP send rate is over optimistic. Thus, the PEP element withholds subsequent premature ACKs. The release of these temporarily withheld premature ACKs will be clocked by the arrivals of actual ACKs. This way, the PEP element can use TCP sender’s self-clocking to control its send rate.

In some sense, the watermark sets the optimal rate at which the active network element would expect the sender to inject packets into the network. To base this optimal rate on the bandwidth currently available to the TCP flow, the watermark should be dynamical and not fixed. A fixed watermark threshold to decide to generate premature ACKs or not yields sub-optimal performance or serious congestion. For example, assuming that the buffer length is below the watermark and the available bandwidth goes down, with a fixed watermark the PEP element would still send out premature ACKs to cause the sender to send more data. This could lead to buffer overflow and congestion. On the other hand, if the available bandwidth goes up but the watermark does not change accordingly, the PEP element may withhold premature ACKs although the network can transmit more, resulting in sub-optimal performance.

To summarize this architecture, the PEP congestion control augments TCP’s end-point congestion control. When congestion happens in a TCP-unfriendly network, it can take a long time for the end points to notice such changes in the available bandwidth. The PEP congestion control attempts to avoid this problem by shortening TCP’s feedback control loop with premature ACKs, and by feeding a more accurate indication with the control of the watermark. The watermark can further alleviate congestion by delaying ACKs hence slowing down flows.

IV. Simulation Results

We have implemented our PEP architecture in NS2. The simulation setup is the same as in Section II-A, except that a PEP agent is place at node C (see Fig. 1). C is the ingress edge router for the simulated heterogeneous network. After we collected the data for the new simulation runs, we compare them with the previous results (when there was no PEP). We have plotted the data in a series of charts. When a curve is labeled “TCP” in the charts, it presents the data from the previous simulations; when a curve is labeled with “PEP”, it presents the data from the new simulations with PEP elements in place.

A. *Slow Start*

We first investigate the effect of PEP during the slow start period. We use constant bandwidth for this study and focus on the first few seconds of each TCP flow. Fig. 5 compares the base bandwidths between regular TCP and TCP flows passing through a PEP element. The results show that PEP gives us a higher throughput with 150% improvement over TCP. This is primarily due to a quicker slow-start. In the simulation network, the PEP element receives data segments from the sender in 10ms. So for every 20ms, a premature ACK can reach back the sender to double its congestion window, while normally TCP would take 220ms (if d=100ms) or 420ms (if d=200ms) to receive the first ACK and to double the congestion window. Therefore, PEP avoids the problem of long slow-start period for networks with TCP-unfriendly characteristics.

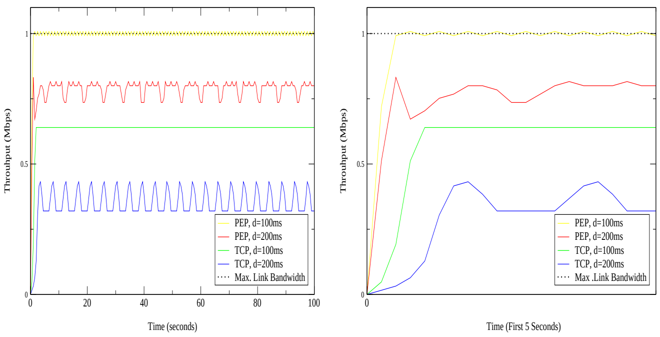
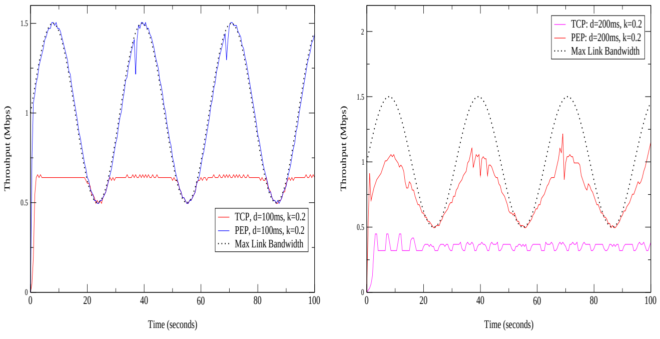


Fig. 5. Throughput Comparison between TCP and PEP

B. Variable Bandwidth

We then investigate the effect of PEP on TCP's performance under bandwidth oscillations. We choose k=0.2 as the angular frequency in our sine oscillation function, as it is realistic to have around 30 seconds per oscillation cycle. Fig. 6 compares the TCP throughput without PEP and with PEP, when the network delay is 100ms or 200ms.



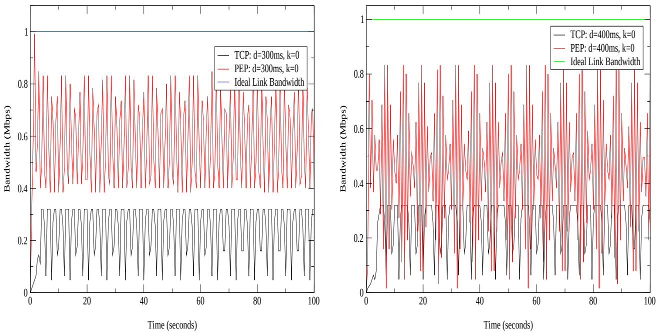
(a) d=100ms (b) d=200ms

Fig. 6. Throughput Comparison under Variable Bandwidth

The figures show that when the bandwidth reaches its trough, the throughput for both PEP and TCP cases is low. However, when the bandwidth goes up, TCP without PEP fails to adjust its rate quick enough and its performance suffers. On the other hand, with the help for PEP, TCP can catch up the bandwidth increase and the throughput is improved. This shows that, by adjusting the watermark accordingly, the PEP element can improve the performance of a TCP flow even though the network has TCP-unfriendly characteristics.

C. Very High Delays

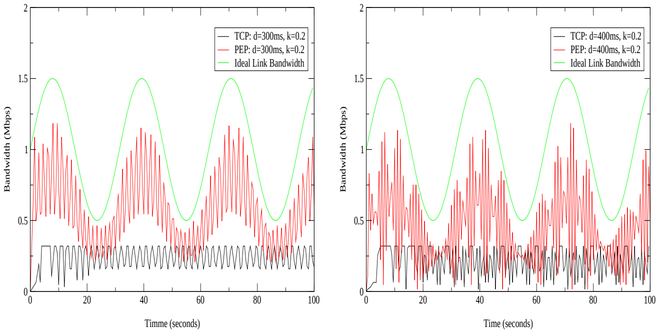
Next, we increase the bandwidth d to 300ms and 400ms to study the effect of very high delays. The results are drawn in Fig. 7. Similarly to the previous case, TCP performs worse without PEP than with PEP. The reason is similar too: standard TCP cannot predetermine the high delays beforehand and the sending rate is low even though there is enough bandwidth.



(a) d=300ms (b) d=400ms

Fig. 7. Throughput Comparison under Very High Delays

When the bandwidth is varied sinusoidally (see Fig. 8), TCP cannot utilize the bandwidth when it goes up. Essentially, what the TCP sender receives from the ACK clocking is the average bandwidth during a round-trip time. Coupled with the fact that TCP has a limited buffer space, it tends to slow down its sending rate. PEP, on the other hand, uses the dynamic watermark mechanism described in previous section to ensure that the ACK timing reflects the actual bandwidth available. The chart shows that the watermark technique succeeds in smoothing the sending rate to match the bandwidth variation.

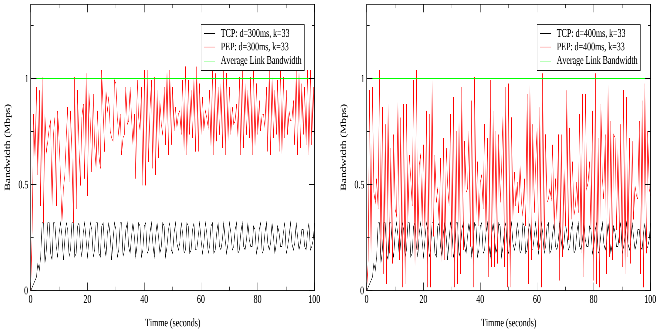


(a) d=300ms (b) d=400ms

Fig. 8. Comparison under Very High Delays and Bandwidth Variation

D. Very Fast Bandwidth Oscillation

Finally, we vary the frequency of sinusoidal bandwidth from k=0.2 to k=33, which corresponds to an oscillation period of around 200ms. Fig. 9 compares the throughput measured under this setup. It shows that PEP's performance improvements are even higher. This further proves that PEP can correct TCP's lack of responsiveness to fast bandwidth oscillation.



(a) d=300ms (b) d=400ms

Fig. 9. Throughput Comparison under Very High Bandwidth Variation

V. Conclusion and Discussion

In this paper, we have first shown that TCP performance falls short of the ideal when the delays are large and the available bandwidth varies with time. We then introduced a new PEP-based architecture to tackle the problems. Our architecture use premature-ACK and dynamic watermark mechanisms to ensure a rate based flow between the sender and receiver, and to modulate this rate by the variance of the dynamically varying bandwidth of the network. Through simulation experiments, we have shown that our PEP architecture makes TCP flows more adaptable to long delays and variable bandwidth. In many cases, the performance of these PEP-assisted flows matches with the calculated levels.

While work has been done elsewhere to improve slow start and congestion avoidance mechanisms to deal with the long delay problems (see reference [6] for a good survey), there is little work that addresses the variable bandwidth problems directly. Our work is among the first attempts through an architectural solution. Further, it is innovative to use the adaptive watermark mechanism for TCP congestion control.

Our approach has deviated from the traditional approaches to the TCP performance problems, which often require changes to TCP end-points by adopting either a new TCP mechanism (e.g. TCP extensions and new algorithms as described in literature [7], [8], [9], [10]) or a non-standard configuration (e.g. with extraordinary large window size). There is no doubt that these solutions can improve the TCP performance, but the biggest disadvantage is that we would need to ascertain the state of the intermediate networks at the outset to decide upon an appropriate configuration. Furthermore, changing TCP at the end-points is not only somehow unrealistic (it would require upgrading almost all the computers in the world), but also very risky. Despite its ineffectiveness in networks with unfriendly conditions, the current TCP congestion control has been proven, both theoretically and through 20+ years of practice on the Internet, to be extremely robust and resilient. Since changing that may cause unexpected consequences like congestion collapse, we had chosen a localized solution – one that strictly limits the changes to those TCP-unfriendly networks themselves.

While many have criticized the PEP architecture for violating the end-to-end principle [1], we argue that, although it is more preferable to use link layer mechanisms to correct the TCP-unfriendly characteristics in a TCP-unfriendly network, there exist certain environments where such compensation is not available or too costly. The two network conditions that we are addressing in this paper, networks with long delays and networks with highly volatile bandwidth, are two such examples. Moreover, PEP have been proven practical and effective for other link characteristics as well, such as the TCP snooping technique for lossy wireless links [11] and the protocol booster idea[12]. Compared to snoop, our PEP does elaborate congestion control.

One of our conjectures is that with our proxy, we can provide better QoS with better jitter and delay guarantees by fine tuning the watermark. In other words the watermark would become a function of QoS requirements like jitter, bounds on delay and threshold bandwidths. Such proxies would become invaluable in QoS networks such as diffserv. That is one of the future directions of this work.

Above all, our approach has achieved the goal of improving TCP performance for TCP-unfriendly networks. Besides, with the potential wide-spread of active control and management elements inside the networks, our architecture will coexist harmoniously with them and help managing traffic without sacrificing the performance.

References

[1] J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-To-End arguments in system design," *Proceedings of the 2nd International Conference on Distributed Systems*, pp. 509–512, April 1981.

[2] L. Breslau et.al., "Advances in network simulation," *IEEE Computer, 33 (5), pp. 59-67*, May, 2000.

[3] L.L. Peterson and B.S. Davie, *Computer Networks, A Systems Approach 2e*, Morgan Kaufmann, 1999.

[4] J. Border et. al., "Performance enhancing proxies," RFC 3135, http://www.ietf.org/rfc/rfc3135.txt, July 2000.

[5] R. Braden (Ed.), L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "Resource ReSerVation Protocol (RSVP) – version 1 functional specification," IETF RFC 2207, Sept. 1997.

[6] M. Allman et.al., "Ongoing TCP research related to satellites," RFC 2760, Feb. 2000.

[7] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," IETF RFC 1323, May 1992.

[8] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithms," IETF RFC 2582, Apr. 1999.

[9] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *ACM SIGCOMM'94*, pp. 24–35, May 1994.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *ACM SIGCOMM'2000*, Aug. 2000.

[11] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP performance over wireless networks," *ACM SIGCOMM'96*, Aug. 1996.

[12] D. Feldmeier et. al., "Protocol boosters," *IEEE Journal On Selected Areas in Communications, 16(3)*, April 1998.