Design and Implementation of Split TCP in the Linux Kernel

Rahul Jain, Teunis J. Ott

{rbj2, ott}@NJIT.EDU
Computer Science Department
New Jersey Institute of Technology

Abstract

With all its history of reliable performance, the TCP protocol is known to break down in a situation with high loss and high RTT. A known solution is "Split TCP", where one or a few proxies (helper boxes) are used to break the end-to-end TCP connection into a few (almost) independent legs. Each leg has its own feedback, congestion control, error control mechanism, etc.

The main contributions of our work are the design and implementation of "Split TCP" using Netfilter Hooks in the Linux kernel, and the use of IP over IP for transport. The kernel implementation reduces overhead. The implementation used leaves TCP packets and flags intact, thus allowing use of Telnet (etc) over a Split TCP connection. The primary area of use is for Internet connections, irrespective of the user application. Connections can be split into legs having high RTT or high loss, preferably not both. The use of IP over IP allows use of many helper boxes in a connection and makes it easier to achieve transparency for the original end–hosts.

I. Introduction

Over the years, the Transmission Control Protocol (TCP) has been the widely accepted means of communicating between computers. Though in most situations TCP offers reliable performance, there are known network conditions, mainly a combination of high Round Trip Time (RTT) and high loss or even fading, that drastically affect its performance. The "Square Root Law" [OKM96], [PFTK98] gives a theoretical argument. The measurements from the PingER project [Pin] of the Internet End-to-end Performance Monitoring (IEPM) group and [MFR02] confirm the theoretical insight. We used results from [Pin] in our choice of round trip times and delays in our investigations of actual performance.

Measurements and theory indicate that the larger the RTT and the probability of loss, the lower the TCP throughput. The situation of interest in our work is that with a high quality, high RTT leg, say one containing a satellite link between the USA and a third world country, and a low quality, low RTT, high drop (and possibly even fading) leg in the third world country.

In this paper we study and enhance "Split TCP" as a solution for the above problem. We describe the design and implementation of "Split TCP" in the networking stack of the Linux kernel.

Split TCP has been extensively researched [BSAK95], [BB95], [BS97], [HA97], [KKFT02] and shown to improve the performance of TCP in Mobile Networks. It works by breaking the end-to-end TCP connection into two or more "legs". In this paper the term "leg" is used to describe a path between two computers that may go through several routers.

For reasons that will become clear in Section II we consider our Helper Boxes (HBs) as specialized routers. Fig. 1 shows a Split TCP mechanism with one HB. A HB acts as a proxy for D (destination) while talking with S (Source) and acts as a proxy for S while talking with D. This is achieved through Acknowledgement Spoofing and maintaining TCP parameters at the HB that are essential for the flow control of each leg. It also maintains, for each of its legs, parameters for RTT estimation, error control and congestion control mechanism. By splitting the connection, each of the legs now has fewer network problems to deal with than the end-to-end connection. This results in an overall increase in the TCP performance of the split connection. We mathematically prove this in Section IV. A property of our design is that Split TCP is completely transparent, hence not requiring any modifications in the networking code of the end hosts.

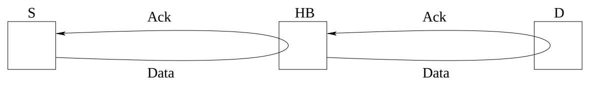

Fig. 1. Split TCP connection

The rest of the paper is organized as follows. In Section II, we outline the network environment behind the design of our form of Split TCP. Section III discusses the design of our version of Split TCP in the Helper Box. We give a mathematical proof of why Split TCP will improve the TCP performance in Section IV. We present the results of our experiments in Section V. In Section VI we compare our work with the previous research contributions. We provide the conclusions of our work in Section VII.

II. Split TCP Design Environment

We have designed our form of Split TCP primarily for the following situation: There is a "campus A" in the USA and a "campus B" in an underdeveloped country, say in Africa. There is a leg from "reasonably close to campus A" to "reasonably close to campus B" that has high RTT and low loss. This leg could contain a satellite link. From (say) the satellite earthstation on there is a leg of questionable quality through the third world country.

The theory of Section IV shows that in a one HB situation the optimal location for that HB is the place where the clean, large RTT leg interfaces with the high loss, low RTT leg in the underdeveloped country. One can think of that as in or close to the satellite earthstation in the underdeveloped country. However, this leaves a significant problem: How do we guarantee that all traffic between campus A and B is intercepted by the HB, and that only traffic that is intended to be intercepted is indeed intercepted.

Our solution is as follows: Place a second HB (HB_A) in campus A and a third HB (HB_B) in campus B. Call the HB at the distant earthstation HB_I (I for intermediate). Give campus A a network address a.b.c.d/n and give campus B a network address w.x.y.z/m.

In campus A, route all traffic to w.x.y.z/m to HB_A. In HB_A, embed all packets to w.x.y.z/m in an IP packet (IP over IP) with destination address HB_I. In HB_I, decapsulate the packet and encapsulate the original packet in an IP packet with destination HB_B. At HB_B, take out the original packet and forward to the actual destination. Traffic in the opposite direction is handled similarly.

This mechanism has the advantage that the original end hosts do not need any modifications, while at the same time between the HB's we can use modified versions of the TCP feedback mechanisms (which in our situation is implemented at the IP layer in the Linux kernel). This design will work for any number of HB's and "campuses". Each HB has a table mapping the destination address into either the "next HB address" or "to original destination".

Apart from the HB's, the only changes necessary are in the routers of the campuses A and B.

III. Helper Box Design

This section outlines the kernel level design of our version of Split TCP. The Split TCP software resides at the Network layer in the networking stack of the HB's. It was implemented as a Linux Kernel Loadable Module (LKLM) and is registered at the Netfilter hook, NF_IP_PRE_ROUTING [WPR+05].

A kernel level design, although difficult, is more efficient and is what we chose. It works with packets as opposed to byte streams. There is no need for repacketization of the data, saving CPU time. It also helps, at no extra effort, in preserving the TCP flags in a TCP packet. This approach also preserves the packet boundaries between different legs. This is important, for example, when the URG or PSH flag is set. A disadvantage of the kernel level design is the buffer space, which is limited by the maximum main memory of the HB. The buffer space is used by the HB to store the data packets and bookkeeping information about the flows.

The design chosen frees the end hosts of any required code changes and ensures minimal configuration changes (if any) in the end hosts and the networks they belong to. For the HB to function properly, all packets of a flow between the end hosts must be routed through the HB. This helps the HB to maintain an accurate state of the TCP flow. The HB makes use of IP over IP (as explained in the previous section) to communicate with other HB to guarantee this.

The Split TCP implementation in the HB intercepts all network traffic passing through it. An IP packet whose source and destination IP address belong to the special IP address pools of the campuses is picked for special processing. The HB simply routes all other packets (if seen at all) as if it were a regular router. This ensures that all traffic other than that requiring special processing does not get dropped by the HB. In other words, the HB's are designed to be dual functional thus reducing the need for an additional network component.

For each TCP flow that the HB splits, it sets aside a pair of data queues. There is one queue per direction of data flow. These queues are used to cache the data packets in the respective directions. This approach of buffering helps in possible retransmission and data rate mismatch if any, thus isolating the network problems (like drop, delay) of one leg from the other. At any time, a queue will contain data packets that have arrived but have not been forwarded yet, as well as packets that have been forwarded but not acknowledged yet by the next HB (or destination).

The HB also maintains the state of each direction of flow. This is essential for the accurate operation of the HB. It maintains the variables proposed in [Pos81] and more. In addition, a flow tracker is used by the HB to track the split connections between itself and the end hosts. The data queues and the state keepers are a part of the flow tracker.

In our design, the TCP flow connection establishment is end-to-end. This frees the HB from providing any application specific user authentication service. It updates the MSS option of the SYN packets, as necessary, to reflect the use of IP over IP or other TCP options being negotiated. This helps in preserving the packet boundaries from leg to leg ensuring faithful transportation of the TCP flags (URG, PSH etc).

IP packets between the two campuses that are not TCP packets are encapsulated (IP over IP) but no state is maintained.

An important feature that needed to be designed for the HB was Error Recovery. Since the HB acts as a proxy for both the source and the destination host (or previous and next HB, etc), it simulates the error recovery mechanism of both. When acting as a source, HB maintains a RTT estimate and RTO timer between itself and the destination host (or next HB). This helps in the retransmission of a lost packet and entrance into congestion avoidance phase. The TCP NewReno algorithm was implemented to handle the congestion avoidance phase of the HB.

However, there can be packet loss in the leg between the source host (or previous HB) and the HB. The HB has the ability to accept packets after the lost packet. Just like a destination host would, it keeps sending duplicate acknowledgements until it receives the lost packet that increases the acknowledgement number. It also takes care not to forward any data packet after the lost packet. Thus, a HB is designed to re-order the data packets, transmit them in order and retransmit as needed.

Recently we added the window scaling option within the HB. Each HB is designed to use the window scaling option by default. Hence our design supports the use of the window scaling option between the HB's irrespective of whether the end hosts use it or not.

IV. Does Split TCP improve performance?

If all is well for a TCP connection, it will not significantly benefit from a HB. However if the TCP connection has the problem that is being addressed, HB's will surely help. Consider the scenario in Fig. 2 to prove this statement.

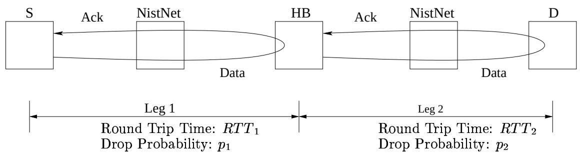

Fig. 2. Split TCP with loss and delay

In the scenario depicted, there is one HB between the two end hosts. To introduce delay and loss in our measurement runs, we added two more computers running NistNet [CS03] in between S (Source) and HB, and in between HB and D (Destination).

The leg between S and HB, "Leg 1" has low loss with drop probability p_1 and a large RTT, RTT_1. This leg is similar to that of a leg having a satellite link. The leg between HB and D, "Leg 2" has high loss with drop probability p_2 and a short RTT, RTT_2. This leg is similar to the final 100 or so miles of the TCP connection in an "underdeveloped country".

The throughput, Thp, under many circumstances, for a TCP connection is given as [OKM96], [PFTK98]

$$Thp \sim \frac{cwnd}{RTT} \sim \frac{MSS}{RTT * \sqrt{p}} \tag{1}$$

where cwnd: congestion window
RTT: Round Trip Time
MSS: Maximum Segment Size
p: Probability of packet loss

This is true as long as the source host has plenty of data packets to send and the congestion window is the only limit on the packets in flight, and the probability, p, is not too large.

Hence, the throughput, for Leg 1 would be

$$Thp_1 \sim \frac{MSS}{RTT_1 * \sqrt{p_1}} \tag{2}$$

as long as the buffer in HB never fills. Similarly, the throughput for Leg 2 would be

$$Thp_2 \sim \frac{MSS}{RTT_2 * \sqrt{p_2}} \tag{3}$$

as long as the buffer in HB never empties.

If the drop probabilities p_1 and p_2 are independent of each other, the total drop probability of the entire route is

$$p_1 + p_2 - p_1 * p_2 \tag{4}$$

As long as at least one of p_1 and p_2 is small, we have $p_1 * p_2 \ll p_1 + p_2$. Hence we can write (4) as

$$p_1 + p_2 - p_1 * p_2 \sim p_1 + p_2 \tag{5}$$

Hence, if there were no HB in Fig. 2, the end to end throughput would be

$$Thp \sim \frac{MSS}{(RTT_1 + RTT_2) * \sqrt{p_1 + p_2}} \tag{6}$$

However, in the situation as shown in Fig. 2, if either one of Thp_1 and Thp_2 is considerably larger than the other or if the buffer in HB is quite large, the effective end-to-end throughput would be

$$\begin{aligned} Eff \cdot Thp &= min\left(\frac{MSS}{RTT_1 * \sqrt{p_1}}, \frac{MSS}{RTT_2 * \sqrt{p_2}}\right) \\ &> \frac{MSS}{(RTT_1 + RTT_2) * \sqrt{p_1 + p_2}}, \end{aligned} \tag{7}$$

so an improvement is (mathematically) assured. The improvement is more pronounced in the situation described, with

$$RTT_2 \ll RTT_1 \sim RTT_1 + RTT_2 \tag{8}$$

and

$$p_1 \ll p_2 \sim p_1 + p_2 \tag{9}$$

(for example, a clear satellite link to an underdeveloped country followed by a high loss link within that country). This explanation shows that the "one helper box" solution requires the HB to be located in the underdeveloped country, but "attached to the clean leg".

Hence we can conclude that using Split TCP for TCP connections having large RTT and high drop probability will result in greater performance. We emphasize here that the throughput of (7) holds when the HB has a large buffer. There might be situations where both the legs are fading but at non-overlapping intervals. In that situation an adequately large buffer is expected to make a large difference.

V. Results

We tested our version of Split TCP against two modes of operation - bulk transfer and real time user response. Telnet and ssh were used to test the real time response while files, of varying sizes, were transferred for the bulk mode. Most of our measurements correspond to the bulk transfer. We mention our findings regarding telnet and ssh towards the end of this section.

The implementation of Split TCP was done on desktop computers consisting of either a Pentium 4, 2.4 GHz processor or an AMD Athlon XP 1700+/2000+ processor. The Linux kernel version on these computers was either 2.4.20 or 2.6.5.

These were the HB's. All computers had 512 MB of RAM. 100 Mbps ethernet links were used for communication.

We measured the performance of Split TCP in the situation of Figure 2, for a few sets of parameter values. We did some measurements with $RTT_2 >> RTT_1$ and p_1 and p_2 of the same order of magnitude. For that situation Section IV predicts that Split TCP will do only moderately better than end-to-end TCP. This is indeed what we found. In this paper we report a sample measurement, Fig. 3, for these conditions. Other measurements are not reported but are available on request.

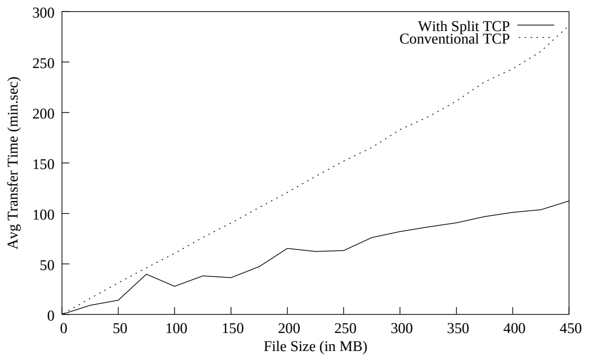

Fig. 3. Measurement for $RTT_2 \gg RTT_1$ & $p_1 = p_2$

For the measurement shown in Fig. 3, the network parameters of Fig. 2 were as follows: $RTT_1 = 10ms$, $p_1 = 5\%$, $RTT_2 = 100ms$ and $p_2 = 5\%$. Various files were transferred between S and D to record our measurements. The file size varied from 25 MB to 450 MB with increments of 25 MB. Each file was transferred 3-5 times to get a good approximation of the transfer time. Each file was transferred under 2 different conditions, one with Split TCP enabled in the HB and one without.

We then changed network conditions to 200ms RTT with 0% drop between S and HB and 10ms RTT and x% drop probability between HB and D, where x varies from 5% to 20% in increments of 1. This resembles the situation involving 1 HB placed near the earthstation of the developing country. A 100MB file was transferred between the end hosts. As seen in Fig. 4, a flow with Split TCP is, on an average, 9.5 times faster than a corresponding regular TCP flow.

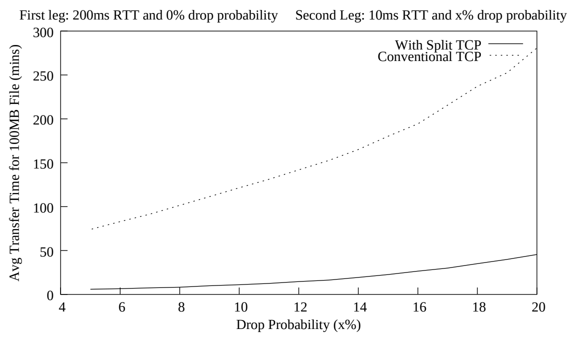

Fig. 4. Varying drop probability in 1 HB setup

We performed the same test on a network setup with 3 HB's as shown in Fig. 5. This closely resembles the network setup discussed in Section II. The drop probability between HB_I and HB_B was varied from 5% to 15% in increments of 1. A 100MB file was transferred between the end hosts. As seen in Fig. 6, a flow with Split TCP is 2.60 to 6.54 (increasing with drop probability) times faster than a corresponding regular TCP flow. We expect a larger difference at higher drop probabilities.

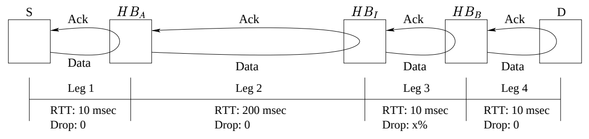

Fig. 5. 3 HB Network Setup

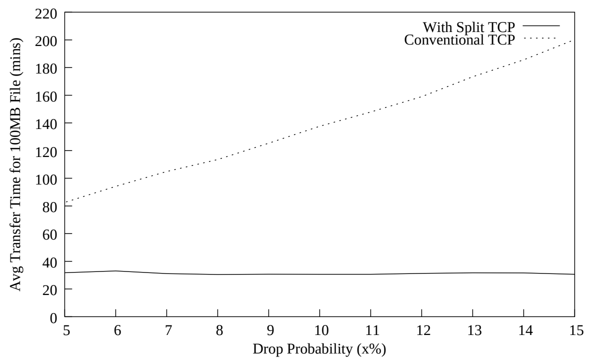

Fig. 6. Varying drop probability in 3 HB setup

For the next experiment, multiple transfers were started simultaneously between the end hosts. This was done to find out the changes in the behaviour of the HB when subjected to more than a single TCP flow. The network setup of Fig. 2 was used for the experiment. The network parameters were as follows: $RTT_1 = 200ms$, $p_1 = 0\%$, $RTT_2 = 10ms$ and $p_2 = 5\%$. A 100 MB file was FTPed between the end hosts. As can be seen from Fig. 7, there is no change in the transfer time of the file even with increasing TCP flows.

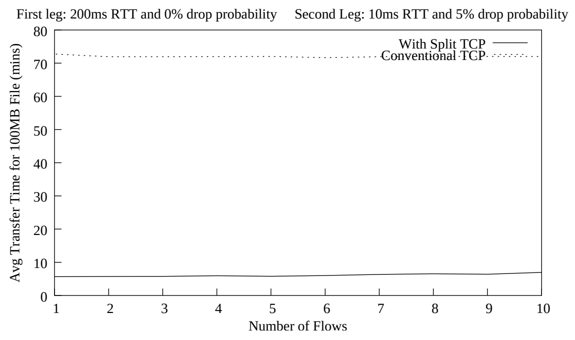

Fig. 7. Transfer times for multiple simultaneous FTP fbws

In all runs reported up till now, we did not use the window scaling option. This limits the effective window to 64,000 bytes instead of the higher congestion window. For the next experiment, we included window scaling in the HB design. The network setup of Fig. 5 was used for this experiment with the drop probability of "Leg 3" fixed at 5%. Various files were transferred between S and D to record our measurements. The file size varied from 25 MB to 250 MB with increments of 25 MB. Each file was transferred 3-5 times to get a good approximation of the transfer time. Each file was transferred under 2 different conditions, one with Split TCP enabled in the HB and window scaling being used only between the HB, and one with Split TCP disabled in the HB and window scaling being used by the end hosts. Fig. 8 compares the average transfer times of the files under the 2 conditions. A flow with Split TCP and window scaling option is on an average 4.99 times faster than a corresponding regular TCP flow with the end hosts using window scaling.

Under the same network conditions (drop probability of 5%), the average transfer time of a 100 MB file with Split TCP and window scaling is twice as fast as the one in Fig. 6. The use of window scaling option by the end hosts, as anticipated, did not change the transfer time of the 100 MB file when regular TCP was used. Hence from Fig. 6 and Fig. 8 we can infer that as the network conditions worsen, the factor of improvement because of Split TCP will increase and the use of window scaling will make this factor even larger.

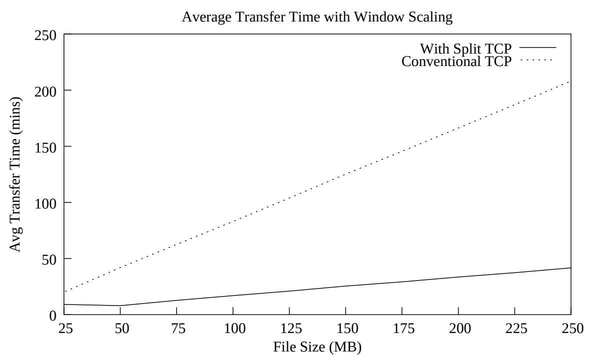

Fig. 8. Average transfer time with window scaling between the HB's

We also verified that with our version of Split TCP it is possible to have a telnet or ssh connection over Split TCP. The successful ssh connections over Split TCP demonstrate that a secure channel is not broken because of the introduction of the HB in the network. As stated earlier, any type of authentication is done end-to-end. For performance with ssh we only have subjective evaluation. With an RTT of 200 msec and a drop probability in the third world leg of 10%, we did not perceive a difference between Split TCP and end-to-end TCP. When the drop probability was increased to 20% we perceived a considerable difference.

In all cases Split TCP gives considerable improvement and the improvement is more significant in the "worst" situation.

VI. Related Work

Over the years, researchers have presented work that highlighted the advantages of using Split TCP in Mobile Networks. In a mobile network, the network connection between the Fixed Host (FH) and the Mobile Host (MH) is broken down into 2 connections at the base station (also know as Mobile Support Routers, MSR). In a mobile network, a packet drop is not always an indication of congestion. For example, environmental factors may cause packet drops. Irrespective of the cause, TCP will initiate the congestion avoidance phase thus reducing its current transmission rate and overall performance.

Most of the related work has concentrated on preventing the sender side TCP from invoking its congestion control for every packet drop. The problems of the wireless connection are separated from the wired connection by splitting the connection at the MSR. Most of the methods introduce a new protocol at the MSR and few changes in the MH. For example, in MTCP [YB94], a new session layer protocol is proposed. While I-TCP [BB95] proposes to introduce I-TCP library in the MH to facilitate its communication with the MSR. And Mobile-TCP [HA97] proposed to introduce a new protocol over the wireless link of the connection, thus allowing the MSR to relieve the MH from buffer and timer management mechanisms. M-TCP [BS97] proposed a restructuring the mobile network into a 3-layer architecture to solve the problem of frequent cell exchanges in addition to the bit-error rate of the wireless link. Lastly, the Snoop protocol [BSAK95] proposed caching data packets at the MSR to increases the end-to-end TCP performance. However not all data packet are cached causing occasional situations when the sender would need to retransmit.

An aspect of most of the above protocols is that they have concentrated on the network problems due to the wireless link. Though these problems do affect the overall performance, the network problems over the wired leg should not be discarded. In our work, we present a Split TCP design irrespective of the networking environment i.e. wired or wireless. Our design does not require any code modifications in the end hosts, thus making it completely transparent.

TCP Splice [MB99], though similar, is not quite the same. TCP Splice concentrates on increasing the performance of the web proxies by relaying the data packets at the kernel level as opposed to through the user space. The connection setup, including user authentication, between the client and the server is performed by the web proxy, which in our case are done end-to-end. This insures the availability of the end hosts to each other. Secondly, the feedback, congestion control and error control mechanisms are performed by the end hosts. Hence a setup with TCP Splice may still suffer poor performance if the network conditions are as discussed previously. We are proposing the use of Split TCP to overcome the network performance by providing each leg with its own feedback, congestion and error control mechanisms.

In [WZZ$^+$06], the authors discuss methods that can be used to find whether Split TCP is being within a network.

According to their findings, 3 cellular service providers (2 CDMA2000 networks and 1 GPRS network) have selectively implemented Split TCP within their networks. Although we believe it to be a kernel level implementation, no details were provided in the paper.

VII. Conclusions

In this paper we presented the design and implementation of Split TCP at the network layer in the Linux networking stack to improve the performance of TCP in an environment having large RTT and high drop probability. Computers called "Helper Box" with modified network stack were introduced in the network. These helper boxes act as performance enhancing proxies. We also proposed how IP over IP can be used to guarantee the flow of traffic through the HB's. In our implementation, the only change required in the network is in the helper box. The routers may require minimal changes in their forwarding table.

We have tested our implementation and found very promising results. With varying drop probability, a connection with 1 HB was 9.5 times faster in comparison than one without. We also experienced a considerable increase in the response time of a telnet and ssh connection over Split TCP under poor network conditions. These results met our expectations of large improvements in situations with worse and asymmetrical drop probabilities.

References

[APS99] M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control.*, April 1999. RFC 2581.

[BB95] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th ICDCS*, pages 136–143, June 1995.

[BKG+01] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations.*, June 2001. RFC 3135.

[BPSK97] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, Vol 5:756–759, December 1997.

[BS97] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communication Review*, 27(5), 1997.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving TCP/IP performance over Wireless Networks. In *ACM Conference on Mobile Computing and Networks*, November 1995.

[CS03] M. Carson and D. Santay. NistNet - a Linux-based Network Emulation Tool. In *Computer Communication Review*, June 2003.

[FHG04] S. Floyd, T. Henderson, and A. Gurtov. *The NewReno Modifications of TCP's Fast Recovery Algorithm.*, April 2004. RFC 3782.

[FMS+98] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. M. Raleigh. Protocol Boosters. *IEEE Journal on Special Aspects of Communication*, 1998.

[HA97] Z. J. Haas and P. Agrawal. Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems. In *Proc. IEEE ICC' 97*, June 1997.

[JBB92] V. Jacobson, R. Braden, and D. Borman. *TCP Extension for High Performance.*, May 1992. RFC 1323.

[KKFT02] S. Koppartyi, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. Split TCP for Mobile Ad Hoc Networks. In *IEEE GLOBECOM*, 2002.

[LE04] M. Liu and N. Ehsan. Modeling TCP Performance with Proxies. *Journal of Computer Communications special issue on Protocol Engineering for Wired and Wireless Networks*, Vol 27:961–975, June 2004.

[MB99] A. D. Maltz and Pravin Bhagwat. TCP Splicing for Application Layer Proxy Performance. *Journal of High Speed Networks*, 8:235–240, 1999.

[MFR02] J. P. Martin-Flatin and S. Ravot. TCP Congestion Control in Fast Long-Distance Networks. Technical report, California Institute of Technology, July 2002. Technical Report CALT-68-2398.

[OKM96] T. Ott, J. H. B. Kemperman, and M. Mathis. The stationary behavior of ideal TCP congestion avoidance. August 1996.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, September 1998.

[Pin] The PingER Project. WWW, `http://www-iepm.slac.stanford.edu/pinger`.

[Pos81] J. Postel. *Transmission Control Protocol.*, September 1981. RFC 793.

[WPR+05] K. Wehrle, F. Pählke, H. Ritter, D. Müller, and M. Bechler. *The Linux Networking Architecture.* Prentice Hall, 2005.

[WZZ+06] W. Wei, C. Zhang, H. Zang, J. Kurose, and D. Towsley. Inference and Evaluation of Split-connection Approaches in Cellular Data Networks. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2006.

[YB94] R. Yavatkar and N. Bhagawat. Improving End-to-End Performance of TCP over Mobile Iinternetworks. In *Mobile 94 Workshop Mobile Computing Syst Appl*, December 1994.