

Utilizing Characteristics of Last Link to Improve TCP Performance*

Wu Xiuchao Indradeep Biswas Chan Mun Choon A.L. Ananda

School of Computing, National University of Singapore
#3, Science Drive 2, Singapore 117543
Email: {wuxiucha, indradee, chanmc, ananda}@comp.nus.edu.sg

Extended Version of the paper published in IPCCC 2005.

Abstract

TCP, perhaps the most widely used transport protocol, was designed for wired links and stationary hosts. But more and more links with different characteristics are used to access the Internet. The variations in link characteristics present different problems to TCP and enhancement of TCP performance over these links remains an active area of research.

Since the current core network of the Internet is composed mainly of optical fiber links and high speed routers, the characteristics of the access link (or the last link) usually dominate the characteristics of the end-to-end network path. As a result, if the client can acquire the characteristics of the access link (e.g. through measurement), the TCP server can utilize heuristics that exploit the access link's characteristics in order to improve the performance of TCP.

In this paper, we propose and evaluate the performance of several heuristics that exploit knowledge of access link characteristics such as bandwidth and packet error rate. Simulation results show that these heuristics can indeed improve TCP performance significantly and demonstrate the potential benefits of using heuristics that adjust TCP congestion control behaviors to match the access link characteristics.

1 Introduction

In recent years, more and more tasks of daily life and business are being carried out over the Internet. TCP [1] is used by most of the popular network applications, such as FTP, Telnet, and WWW.

Currently, the network path used by a TCP connection can be divided into two parts, core network and access network. Core network is comprised of optical fiber links and high speed routers. Many servers

are connected to core network by high speed and reliable links. Access network connects users to core network so that users can access TCP servers. Different communication links (dial-up modem, ADSL, WLAN, GPRS, etc.) can be used as access links. These access links have different characteristics and bring many problems to TCP.

1.1 TCP Protocol

In 1980s, TCP was designed for highly reliable links and stationary hosts. It provides a connection-oriented end-to-end service and ensures the reliable and in-order transfer of data.

TCP protocol uses sliding window mechanism to transmit data. An advertisement window (WND) that states the size of receiver's buffer is used to carry out flow control. TCP protocol also uses congestion control to avoid global congestion collapse. In congestion control, TCP sender probes the highest possible data rate by increasing Cwnd continuously and recovers from congestion by decreasing Cwnd when congestion is detected. The sending window is the smaller one of Cwnd and WND.

Congestion control affects TCP performance so prominently that huge number of works focus on congestion control to improve TCP performance. Thus, approved TCP implementations (Tahoe, Reno, New Reno [2]) differ mainly in their congestion control mechanisms. In the following paragraphs, we present the congestion control of TCP New Reno, the latest approved TCP implementation.

In order to probe for available bandwidth quickly and avoid frequent congestion, TCP sender has two states, Slow Start (SS) and Congestion Avoidance (CA) [3]. In SS state, Cwnd is increased exponentially so that TCP sender can achieve high sending rate quickly. In CA state, Cwnd is increased linearly so that the sender can probe the network and increase

*This paper is based upon work supported by University Research Committee, National University of Singapore under grant R-252-000-203-112.

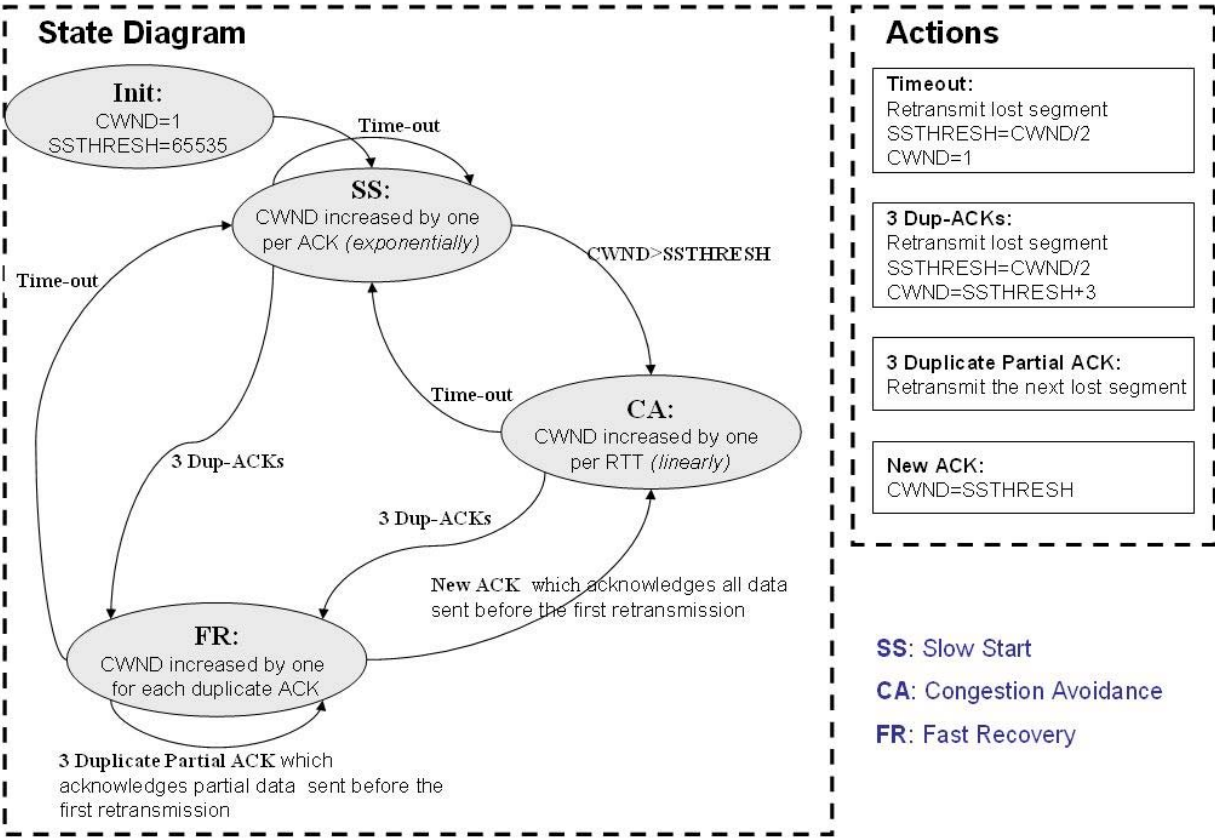


Figure 1: Congestion Control of TCP New Reno

the sending rate gradually without causing congestion too frequently. A variable, Slow Start Threshold (SSThresh), is maintained to determine the state of a TCP sender. When Cwnd is less than SSThresh, the sender is in SS state. Otherwise, it is in CA state.

TCP sender regards segment loss as a congestion signal and detects segment loss by timeout or 3 dup-ACKs. When a segment loss is detected, SSThresh is always set to half of the current Cwnd. If the loss is detected by timeout, Cwnd is set to one, TCP sender enters into SS state, and all unacknowledged segments are retransmitted. If the loss is detected by 3 dup-ACKs, TCP sender retransmits the lost segment, Cwnd is set to SSThresh plus 3, and TCP sender enters into Fast Recovery (FR) state.

In FR state, Cwnd is increased by one segment for each duplicate ACK. When 3 duplicate partial ACKs, which acknowledges partial data sent before the first retransmission, are received, TCP sender just retransmits this new lost segment and Cwnd will not be reduced further. When new ACK, which acknowledges all data sent before the first retransmission, is received, Cwnd is set to SSThresh and the sender returns back to CA state.

The congestion control of TCP New Reno is very complex. Figure 1 presents its state transition diagram.

1.2 TCP and Access Links

Since core network of the Internet is composed of high reliable optical fiber links and high speed routers, the access link (the last link) normally determines the characteristics of a network path used by a TCP connection. In this scenario, the access link is normally the bottleneck link and packet corruption is mostly like to occur over the access link.

With the deployment of different access links on the Internet, TCP faces many problems. For example, TCP regards the corruption over lossy link as congestion and reduces its sending rate unnecessarily, resulting in poor throughput.

TCP performance enhancement for different access links is an active research area. Several TCP implementations (Tahoe, Reno, New Reno) have been approved to handle characteristics of these links. For example, New Reno improves Fast Recovery introduced in Reno to solve multiple segment loss in one RTT, which occurs frequently over LFN [4], wireless links, etc. In addition, many other mechanisms have been proposed to solve problems posed by different access links. Dawkins [5] [6] summarizes TCP performance issues over slow links and lossy links. Balakrishnan [7] summarizes TCP performance issues over asymmetric

links. Allman [8] investigates into enhancing TCP performance over satellite link. The 2.5G and 3G wireless links are investigated in [9].

These enhancements mostly investigate the characteristics of a specific access link and propose new TCP algorithms so that TCP can perform better over this link. But TCP server with these algorithms may perform badly with other clients which use different access links. For example, Large Initial Window [10] is proposed for LFN [4] to probe available bandwidth sooner. But when TCP server with Large Initial Window is accessed by a client through an access link with short delay and narrow bandwidth, TCP server with Large Initial Window is too aggressive and will cause multiple segment loss.

One of the motivations for this work is that the client can often infer the characteristics of the last link. For example, it may be able to infer the available bandwidth according to the type of network interface used (e.g. in the case of DSL) or through measurement. In addition, for wireless network interfaces, characteristics like SNR and Packet Error Rate (PER) can also be obtained through passive measurements. Since the last link characteristics dominate the network path behavior, such information can be used by the TCP server to adjust its congestion control mechanism accordingly to increase the throughput.

In this paper, we consider the following scenario. The TCP client, which is connected to the network through the access link, reports the access link characteristics to the TCP server. After receiving this information, the TCP server utilizes heuristics that match the access link characteristics of the TCP client. In this way, the TCP server can simultaneously handle many TCP connections using different access links and selects the heuristics to match the properties of the individual access link.

The paper is organized as follow. In section 2, we highlight some related works. In section 3, we present several heuristics based on bandwidth and PER of the last link. In section 4, we give the simulation results of these heuristics in NS2 and give some analysis for these results. We conclude in section 5.

2 Related Works

The idea of utilizing knowledge of the client had been used in TCP performance enhancement for a long time.

In SACK [11], instead of only sending the next expected sequence number, TCP receiver sends the information of segments received out of order in a TCP option. According to these information, TCP

sender can implement selective retransmission which can avoid unnecessary retransmission. In addition, SACK enables TCP sender to detect multiple segment loss in one RTT and avoid unnecessary Cwnd reduction.

In HACK [12], TCP receiver sends the sequence number of the corrupted segment in the HeAder ChecKsum option (HACK). Normally, the TCP header of a corrupted segment is still correct. TCP receiver lets wireless interface driver send the corrupted segment to TCP layer. Thus, TCP receiver can feed back its sequence number to TCP sender. With this option, TCP sender can distinguish the reason of segment loss — congestion or corruption. Thus, TCP sender can avoid unnecessary reduction of Cwnd.

RCP [13] utilizes the characteristics of the last link to improve TCP performance. Instead of TCP sender, it is TCP receiver that carries out congestion control according to the characteristics of the last link known by TCP receiver. TCP receiver then tells the sender how many packets can be sent. RCP is a good proposal to improve TCP performance over wireless links. But TCP sender totally loses its control to sending rate. This may bring severe problems with cheating TCP receivers.

In this paper, we will let TCP receiver (the client) feed back last link’s characteristics in a TCP option and let TCP sender (the server) utilize these knowledge conservatively to improve TCP performance.

3 Proposed Heuristics

The last or access link is normally the bottleneck link. The last link normally dominates PER of the whole network path. We let TCP receiver feed back the bandwidth and PER of the last link to TCP sender through a TCP option. With these values, TCP sender uses several heuristics to improve TCP performance. In the following sub-sections, we describe three heuristics: Max Cwnd Tuning, Initial Cwnd & SStresh Tuning, and Packet Error Rate (PER) heuristic.

3.1 Max Cwnd Tuning

The bottleneck link of a network path determines the maximum amount of data that can be sent through the path. Once the bottleneck link is fully utilized, pumping in more data will result in packet loss. By default, TCP sender keeps probing for more available bandwidth by increasing its Cwnd until a segment loss is detected. When segment loss is detected, it will reduce Cwnd to one or half of the current value. After that, TCP sender starts probing again. This cycle keeps repeating during the life of a TCP connection.

If the maximum available bandwidth of the bottleneck link is known, we can calculate the maximum value of Cwnd that can be safely used. If TCP sender constrains Cwnd between one and its maximum, TCP sender will not exceed the bottleneck capacity and keep sending at the constant maximum rate in steady state. This will flatten the saw-tooth like Cwnd graph of TCP and improve TCP throughput. In addition, the number of lost packets will be decreased too.

The client, which uses the last link directly, has the ability to obtain last link’s bandwidth. It may get the bandwidth of the last link through the type of currently used network interface. The bandwidth of some access links, such as wireless links, may change due to contention, link quality, and user movement. In this scenario, link characteristics estimation algorithms, such as [14], should be used by the client to measure the available bandwidth. In this case, an initial conservative value can be used while the bandwidth estimation is being updated. Previous bandwidth estimation can be utilized for a new TCP session unless the network interface has been brought down or packet loss exceeds some threshold.

According to above observation, **Max Cwnd Tuning** is designed. In this heuristic, the client reports the bandwidth of the last link to the server. TCP sender then calculates $Cwnd_{max}$ according to the following equation.

$$Cwnd_{max} = \frac{BW_{lastlink} * RTT}{8 * L_{segment}}$$

Here, $L_{segment}$ is the length of TCP segments and $BW_{lastlink}$ is the bandwidth of the last link. During congestion control, TCP sender first compares the current Cwnd with $Cwnd_{max}$ when an ACK is received. If Cwnd is less than $Cwnd_{max}$, it keeps increasing Cwnd linearly or exponentially according to its state. Otherwise, it stops to increase Cwnd.

In this heuristic, TCP sender only sets the maximum of Cwnd. When the last link is the bottleneck, this heuristic could improve the performance of TCP since it smoothens the sawtooth-like Cwnd graph. When other links become the bottleneck link due to cross-traffic, TCP sender still can reduce Cwnd without any restriction from this heuristic. That means, Max Cwnd Tuning does not change the responsiveness and fairness of TCP protocol.

3.2 Initial Cwnd & SStresh Tuning

When a connection is set up, Cwnd is set to one by default. Then Cwnd is increased exponentially to

probe available bandwidth. If the network path is a long fat pipe [4], TCP sender still needs some time to acquire all of the available bandwidth. That means TCP can not fully utilize network capacity during its slow start phase and the throughput is poor.

When a connection is set up, $SSThresh$ is set to 65535 by default. Normally, TCP sender will suffer congestion before $Cwnd$ is increased to 65536. Since $Cwnd$ is increased exponentially, at the end of the first slow start phase, TCP sender sends too many segments, and often results in multiple segment loss. These lost segments waste bandwidth of links even before they arrive at the bottleneck link. In addition, segment loss will cause $Cwnd$ reduction and result in poor throughput.

In order to solve above problems, **Initial Cwnd & SSThresh Tuning** is designed. In this heuristic, we also utilize the bandwidth of the last link, which is normally the bottleneck link. With bandwidth reported by TCP receiver, TCP sender sets $Cwnd_{init}$ and $SSThresh_{init}$ (the initial values of $Cwnd$ and $SSThresh$) according to the following equation.

$$Cwnd_{init} = SSThresh_{init} = \frac{BW_{lastlink} * RTT}{8 * L_{segment}}$$

With Initial $SSThresh$ Tuning, TCP can effectively avoid multiple segment loss that may occur at the end of the first slow start phase. With Initial $Cwnd$ Tuning, TCP sender skips slow start and starts in CA state so that it can achieve high sending rate immediately. Hence, this heuristic can improve TCP performance, especially for short flows.

Due to Initial $Cwnd$ Tuning, this heuristic is more aggressive than the slow start phase of TCP protocol. It may cause segment loss at other routers and adversely affect cross traffic. Note that, the normal congestion control will resume after a packet loss. Hence, Initial $Cwnd$ is only aggressive at the start and may work well for short flows. As for Initial $SSThresh$ Tuning, we can use it safely.

3.3 Packet Error Rate (PER) Heuristic

When a TCP segment is transmitting over a network path, it may be corrupted over a link (corruption) or be dropped at a router (congestion). But TCP assumes all losses are caused by congestion and reduces its $Cwnd$. In high PER environments, the wrong interpretation causes frequent $Cwnd$ reduction and results poor throughput. The challenge lies in correctly identifying the cause of a segment loss, and only decreasing $Cwnd$ when packet is lost due to congestion. Many works have been done to solve this

problem. Balakrishnan [15] gives a good survey of these works.

Within current Internet, PER of a network path is normally determined by the last link. And the client can measure its PER by some algorithms, such as the algorithm in [14] proposed for Wireless LAN. In principle, PER are measured continuously and an initial conservative value should be used while the estimation is being updated. Previously used or current PER can be utilized for a new TCP session unless the network interface has been brought down or packet loss exceeds some threshold.

Based on these facts, PER heuristic is designed to improve TCP performance over lossy links. TCP receiver first measures PER of the last link and reports to the server. TCP sender then uses the following algorithm to improve TCP performance over lossy link.

Whenever a segment loss is detected, TCP sender first calculates current Packet Loss Rate (PLR) experienced by this connection. $PLR = \frac{N_{retr}}{N_{tx}}$. Here, N_{retr} is the number of segments retransmitted on this connection and N_{tx} is the number of segments transmitted on this connection. TCP sender then compares PLR with PER of the last link. If PLR is larger than PER, TCP sender assumes congestion and will do congestion recovery by reducing $Cwnd$. Otherwise, TCP sender only retransmits the lost segment without $Cwnd$ reduction.

While the performance of this heuristic is affected by its accuracy in differentiating wireless loss and congestion loss, high detection accuracy is not necessary for performance improvement. This is due to the fact that whenever the heuristic determines that it is a wireless loss and does not back-off, throughput can be improved. However, false negatives may result in overly aggressive behavior and be unfair to other TCP flows in short term. In the simulation, we will show through many experiments that this heuristic can improve the performance and at the same time be friendly to other TCP flows in long term.

4 Simulation and Analysis

In order to verify the effectiveness of our proposed heuristics, we adopt simulation in this paper. Based on TCP New Reno, we implement these heuristics in NS2 [16]. We then carry out many experiments to test the effectiveness of these heuristics.

In this section, we first discuss the network topology used for simulation experiments. We then present the results of these simulation experiments and give some

analysis of these results.

4.1 Experiment Setup

In the following experiments, we use the network topology shown in Figure 2, which is a typical topology used for investigating TCP performance on Internet.

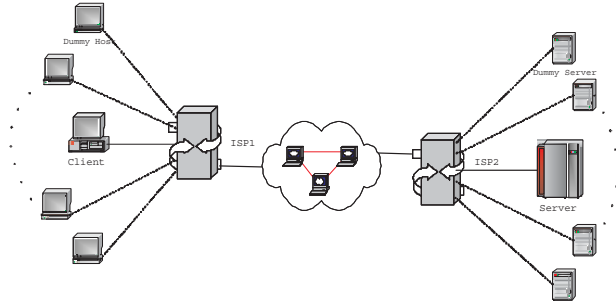


Figure 2: Network Topology

In this topology, the connection between the Client and the Server is that will be investigated. The Client is connected to ISP1 with a 7 Mbps link. The Server is connected to ISP2 with 100 Mbps Ethernet. The two ISPs are connected through the Internet cloud. In fact, we use a 45 Mbps link between the two ISPs to simulate the connection provided by the Internet cloud. The RTT between the two ISPs will be changed in experiments to investigate the performance of our heuristics under different RTTs.

The TCP connection running between the the Client and the Server may use normal TCP New Reno or TCP New Reno with our heuristics. In each experiment, we normally run both TCP New Reno and TCP New Reno with our heuristics to compare their performance. In the following parts, TCP refers to normal TCP New Reno and Heuristic TCP (HTCP) refers to TCP New Reno with some of our heuristics.

In order to make the system close to reality and evaluate the fairness of HTCP, seven dummy nodes are placed on each ISP. They are connected to their corresponding ISP with 100 Mbps Ethernet. Seven TCP New Reno connections are established among the fourteen hosts in order to generate cross-traffic and evaluate the fairness of HTCP. In these seven TCP connections, two connections carry Telnet flows and five connections carry FTP flows. In the following experiments, these flows have a random startup time between 0 and 50 seconds from the start of the simulation. All experiments run for 1000 seconds (15 minutes approximately).

4.2 Max Cwnd Tuning and Initial Cwnd & SSThresh Tuning

In order to verify Max Cwnd Tuning and Initial Cwnd & SSThresh Tuning, we use FTP to measure throughput of the connection between the Client and the Server with different RTT values. For each RTT, we measure throughput of TCP New Reno, HTCP with Max Cwnd Tuning, HTCP with Initial Cwnd & SSThresh Tuning, and HTCP with Max Cwnd and Initial Cwnd & SSThresh Tuning. We then plot these results together to compare their performance.

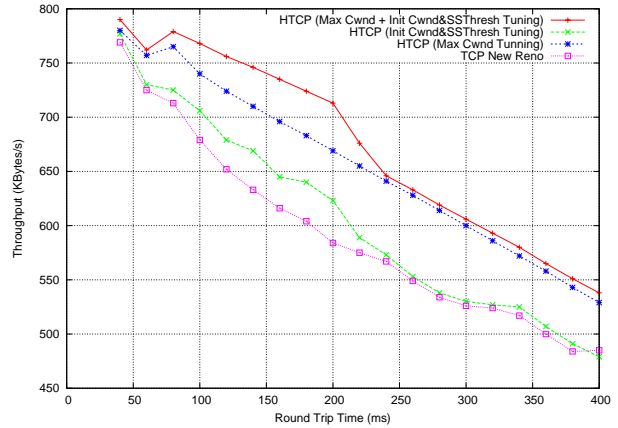


Figure 3: Throughput of TCP and HTCP

Figure 3 presents the result of these experiments. It shows that the performance of HTCP with Initial Cwnd & SSThresh tuning is only a little better than that of TCP New Reno. This is reasonable because Initial Cwnd & SSThresh tuning only affects the start phase of TCP connection. This heuristic works better for short flows. This figure also shows that HTCP with Max Cwnd Tuning performs much better than TCP New Reno. And it shows that TCP with Initial Cwnd & SSThresh and Max Cwnd Tuning performs the best. At the RTT of around 200ms, the throughput of HTCP with Initial Cwnd & SSThresh and Max Cwnd Tuning is nearly 30% higher than that of TCP New Reno.

The above result is quite close to our expectation. Figure 4 provides us an further insight into how these heuristics optimize TCP. TCP New Reno suffers from the under-utilization of slow-start phase and it also periodically suffers from the partial under-utilization as represented by the sawtooth-like Cwnd graph. Using the Initial Cwnd & SSThresh heuristic, we find that the TCP New Reno’s Cwnd graph has been shifted to the left. By setting the Initial Cwnd value with

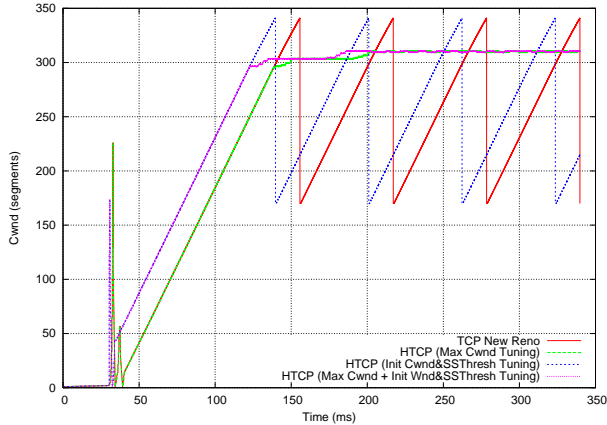


Figure 4: Cwnd Graph of TCP and HTCP

the bandwidth of the last link, we allow TCP sender to reach high utilization earlier than TCP New Reno. This effectively increases the area under the curve, i.e. the throughput of the connection. The curve of HTCP with Max Cwnd heuristic is a relatively smooth line in steady state. This curve causes less segment loss, hence Cwnd reduction. And the area covered under the curve is therefore larger. That means it produces larger throughput. HTCP with Max Cwnd and Initial Cwnd & SSThresh Tuning uses both Initial Cwnd & SSThresh and Max Cwnd Tuning and benefits from both of them. This is the reason why its performance exceeds that of each heuristic applied individually.

According to above simulation results, HTCP with these heuristics does improve TCP throughput. We also know that HTCP with these heuristics keeps the responsiveness and fairness of TCP very well. Thus, HTCP with Max Cwnd and Initial Cwnd & SSThresh Tuning is a good method for TCP performance enhancement. At the same time, we should be cautious to use Initial Cwnd Tuning because of its aggressiveness.

4.3 Packet Error Rate heuristic

In order to verify the effectiveness of PER heuristic in long term, we also use FTP to measure the throughput of the connection between the Client and the Server when PER of the last link varies from 0.08% to 0.8%. We also carry out these experiments under different RTT values. For each pair of PER and RTT, we measure the throughput of TCP New Reno and HTCP with PER heuristic. We then plot these results to compare the performance of TCP New Reno and HTCP with PER heuristic under different PER and RTT.

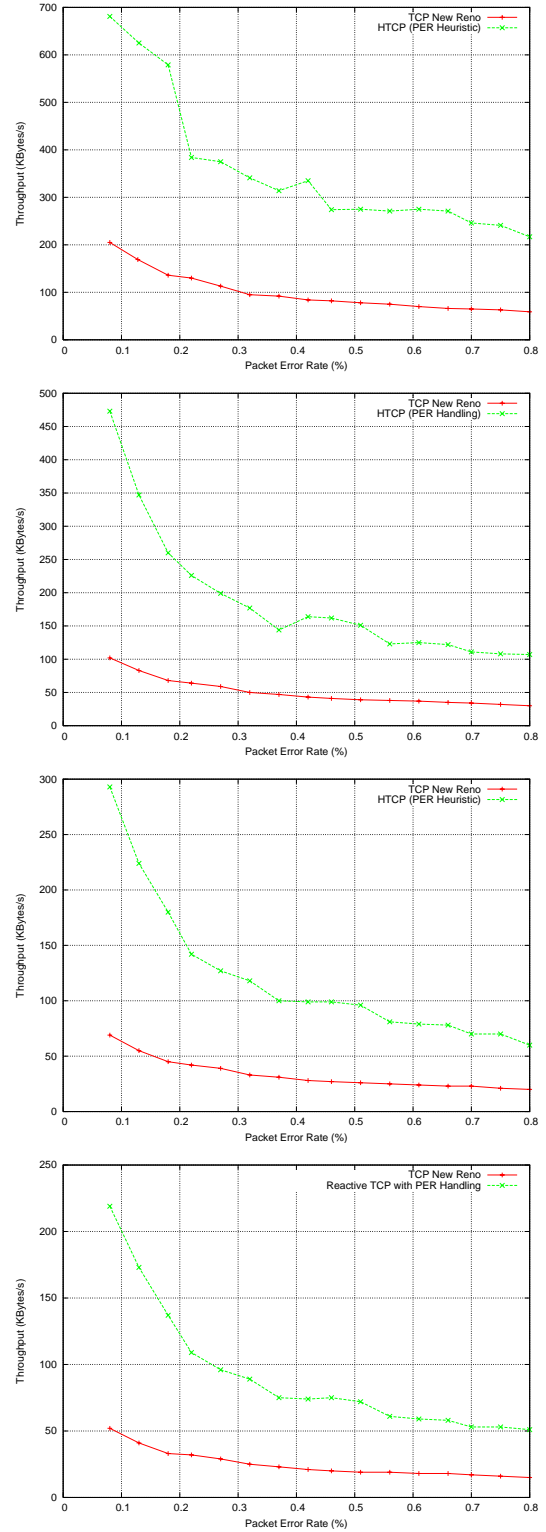


Figure 5: Throughput of HTCP and TCP under Different PER and RTT (100, 200, 300, and 400ms)

Figure 5 presents the results when RTT equals to 100, 200, 300, and 400ms. It shows that HTCP with PER heuristic performs better than TCP New Reno under different RTT and different PER.

RTT	Average Throughput		Dummy Flow Throughput	
	TCP	HTCP	TCP	HTCP
100	103.0	334.9	1370.2	1362.7
200	46.7	168.4	968.2	965.6
300	30.8	108.2	725.8	703.0
400	26.5	92.6	609.6	611.4

Table 1: Statistics for TCP and HTCP (PER)

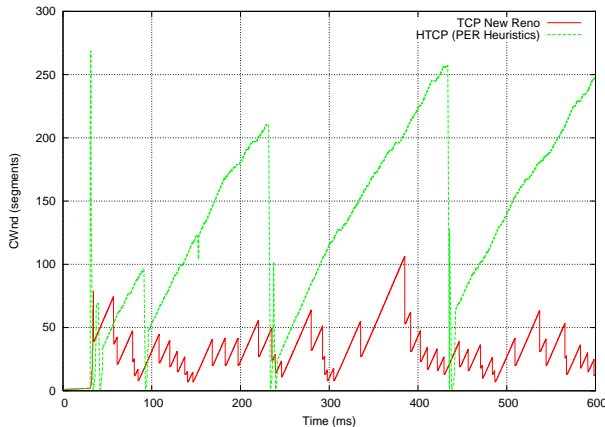


Figure 6: Cwnd Graph for HTCP and TCP

In above experiments, we also calculate average throughput achieved by TCP New Reno and HTCP with PER heuristic. The first two columns of Table 1 shows that the throughput of HTCP connection between the Client and the Server is about three to four times of the throughput of TCP New Reno connection between the Client and the Server. PER heuristic can improve TCP performance quite a lot. The Cwnd graph of HTCP and TCP in Figure 6 shows that HTCP get an opportunity to increase its Cwnd bigger than TCP New Reno. It also shows that the area under the HTCP’s curve is much larger than that of TCP New Reno. That explains the larger throughput achieved by HTCP.

In order to investigate the fairness of HTCP with PER heuristic, we measure the average throughput of five dummy ftp flows which are running simultaneously with the connection between the Client and

the Server. For each pair of RTT and PER, two simulations are executed. In one simulation, the connection between the Client and the Server uses TCP New Reno. In the other one, it uses HTCP with PER heuristic. The last two columns of Table 1 compare these results. It shows that the throughput of dummy flows running with TCP New Reno connection between the Client and the Server is very close to that of dummy flows running with HTCP (PER heuristic) connection between the Client and the Server. That means our PER heuristic keeps the fairness of TCP protocol very well in long term.

According to above results, in long term, PER heuristic can improve TCP performance and keep the fairness.

5 Conclusion

In this paper, we propose three heuristics, Max Cwnd Tuning, Initial Cwnd & SSThresh Tuning, and PER heuristic, that utilize the bandwidth and PER of last link to improve TCP performance.

In this paper, we investigate the effectiveness of these heuristics through simulation. Simulation results show that these heuristics can indeed improve TCP performance significantly and keep the fairness of TCP protocol. These results demonstrate the potential benefits of using heuristics that adjust TCP congestion control behaviors to match the access link characteristics. For example, TCP server adopts Max Cwnd Tuning when the client reports that the bandwidth of last link is small. When the client reports a high PER of its last link, TCP server adopts PER heuristic. Thus, TCP server with heuristics can select proper heuristics according to last link’s characteristics of different clients and simultaneously improve the performance of many clients which may use different access links. Hence, the overall performance of the TCP server can be improved.

There are still a lot of works to investigate into heuristics that utilize characteristics of last link to improve TCP performance. First, access link, such as WLAN, is the bottleneck and the the lossy link simultaneously. And the bandwidth and PER change frequently and abruptly due to contention, link quality, and user movement, etc. We need investigate into the effectiveness of these heuristics over such access links. In addition, we should design and evaluate the performance of new heuristics that utilize other characteristics of the last link, such as asymmetry.

References

- [1] P. J., “Transmission control protocol - darpa internet program protocol specification,” RFC 793,

- DARPA, Sep. 1981.
- [2] S. Floyd and T. Henderson, "The newreno modification to tcp's fast recovery algorithm," RFC 2582, Apr. 1999.
 - [3] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM*, 1988.
 - [4] V. Jacobson, R. Braden, and D. Borman, "Tcp extensions for high performance," RFC 1323, May 1992.
 - [5] S. Dawkins, G. Montenegro, M. Kojo, and V. Magret, "End-to-end performance implications of slow links," RFC 3150, 2001.
 - [6] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "End-to-end performance implications of links with errors," RFC 3155, 2001.
 - [7] H. Balakrishnan, V. N. Padmanabhan, G. Fairhurst, and M. Sooriyabaddara, "Tcp performance implications of network path asymetry," RFC 3449, 2002.
 - [8] M. Allman, D. Glover, and L. Sanchez, "Enhancing tcp over satellite channels using standard mechanisms," RFC 2488, 1999.
 - [9] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, "Tcp over second (2.5g) and third (3g) generation wireless networks," RFC 3481, 2003.
 - [10] M. Allman, S. Floyd, and C. Partridge, "Increasing tcp's initial window," RFC 3390, 2002.
 - [11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "Tcp selective acknowledgment options," RFC 2018, 1996.
 - [12] B. R.K, B.P.Lee, K.R.R.Kumar, L.Jacob, W.K.G.Seah, and A.L.Ananda, "Tcp hack: a mechanism to improve performance over lossy links," *Computer Networks*, vol. 39, pp. 347–361, 2002.
 - [13] H. Hsieh, K. Kim, Y. Zhu, and R. Sivakumar, "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces," in *In Proc. of ACM MOBICOM'03*, Sep. 2003.
 - [14] W. Xiuchao and A. Ananda, "Link characteristics estimation for ieee 802.11 dcf based wlan," in *In Proc. of IEEE LCN'04*, Nov. 2004.
 - [15] 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, 1997.
 - [16] "Ns2 network simulator," available online at <http://www.isi.edu/nsnam/ns/>.