TCP - Congestion Control Algorithm for Enhancing High-Speed Networks

V.Sasirekha , Dr.C.Chandrasekar

Department of MCA, K.S.R College of Engineering,Tiruchengode.

v_sasirekha@hotmail.com

Abstract

Current TCP congestion control can be inefficient and unstable in high-speed wide area networks due to its slow response with a large congestion window. Several congestion control proposals have already been suggested to solve these problems and two properties have been considered: TCP friendliness and scalability, to ensure that a protocol does not take away too much bandwidth from TCP, while utilizing a bandwidth of high speed networks efficiently. In this paper, we propose a new variant of TCP for a high-speed network which combines delay-based congestion control with loss-based congestion control. Our simulation results show that proposed scheme performs better than the existing high-speed TCP protocols in terms of fairness, stability and scalability, while providing friendliness at the same time.

1. Introduction

Congestion control in the Internet was introduced in the late 1980s by Van Jacobson. Jacobson's algorithm which is implemented in the transport layer protocol called TCP (Transmission Control Protocol) has served the Internet well during a time of unprecedented growth. However, the algorithm was designed during a time when the Internet was a relatively small network compared to its size today. Therefore, there has been much interest in reexamining the role of congestion control in the Internet with the goal of enhancing TCP to make it scalable to large networks. We begin this survey with a simple model of a particular version of Jacobson's algorithm called TCP-Reno. We show that the algorithm may not be stable when the network size becomes large, i.e., if the feedback delays are large or the capacity of the network is large. We then present Kelly's model where congestion control is viewed as a distributed control algorithm for achieving fair resource allocation in a network we turn our attention to connection-level models of congestion control. In traditional congestion control models, the number of users on the various routes in the network is assumed to be fixed. In the connection level model, the number of files in the network is time-varying and is described by a Markov chain. Between the time instants when there is a file arrival or departure, it is assumed that congestion control occurs instantaneously. Thus, the connection-level model operates at a slower time-scale compared to the time required for congestion controllers to converge. Under this assumption of time-scale separation, we show that the Kelly model of resource allocation is efficient, i.e., if the total load on each link in the network is less than its capacity, then the network is stable with many users. The remarkable feature of the algorithm is that the complex interactions between the users in the network can be captured using quantities that can be measured easily by each individual user. Next, we study the stability of this congestion control mechanism in the presence of feedback delay. We primarily concentrate on a new technique using Razumikhin's theorem that proves global stability of the congestion controllers under some assumptions on the control parameters. TCP has already been widely adopted as a data transfer protocol for the Internet. The demand for high-speed applications such as bulk-data transfer, multimedia web streaming, high energy and nuclear physics, astronomy, bioinformatics, earth sciences, storage area network, and grid networking has increased.

2 Related Work

The importance of congestion control is now widely acknowledged and extensive research has already been done to enhance the performance of TCP. TCP congestion control is composed of two major algorithms: slow-start and congestion avoidance algorithms which allow TCP to increase the data transmission rate without overwhelming the network. TCP uses a variable called congestion window (*cwnd*) and cannot inject more than *cwnd* segments of unacknowledged data into the network. The TCP congestion avoidance algorithm is called AIMD and it is the basis for steady state congestion control. In the congestion avoidance phase, TCP increases the congestion window by one packet for each RTT and halves the congestion window in the event of a packet loss. TCP congestion control high speed TCP was introduced by S. Floyd in [2] as a modification of the TCP congestion control mechanism, to improve the performance of TCP in fast, long delay networks.

The high speed TCP response function is represented by a new additive increase and multiplicative decrease parameters. These parameters modify both the increase and decrease parameters according to the *cwnd*.

STCP was described by T. Kelly in [3]. Here, the congestion avoidance algorithm of the STCP is MIMD (Multiplicative Increase and Multiplicative Decrease). [5] revealed that HSTCP and STCP have a fairness problem when multiple flows with different RTTs are competing. Also, [5] introduced BIC that attempts to correct the RTT unfairness. BIC regards congestion control as a searching problem in which the system can give binary feedback through packet loss as to whether the current congestion window is larger than the network capacity. BIC uses a binary search scheme to quickly find an estimated equilibrium window size, and then slowly increases the congestion window.Delay-based congestion avoidance protocols attempt to control the congestion window based on RTT measurements.

In addition, the correlation between increased delays (or RTTs) and congestivelosses has recently been challenged [10], thereby raising serious doubtsas to the effectiveness of DCA algorithms given that their main assumption is that RTT measurements can be used to predict and avoid network congestion.

3 High speed TCP Protocol: Mechanisms and Deployment

In this section we propose a new variant of TCP for high-speed networks that provides high utilization, stability, and fairness. In contrast to TCP Vegas, high speed TCP uses effective RTT to avoid the effects of reverse path congestion. And rather than preventing packet loss as in TCP Vegas, for additive increase mechanism, high speed TCP uses a backlog as a binary feedback to determine whether the network is fully utilized. Also, effective RTT is used to refine multiplicative decrease mechanism of HSTCP to achieve high link utilization, while guaranteeing TCP friendliness comparable to that of HSTCP.

3.1 Delay Measurement

If network congestion occurs in the backward path, TCP Vegas-like protocols may overestimate RTT and unnecessarily decrease congestion window. By using the TCP timestamp option, our mechanism obtains samples of queueing delay on the forward and backward paths separately. Note that the sender and receiver clocks do not have to be synchronized since we are only interested in the relative time difference. By distinguishing the direction in which congestion occurs, eHSTCP is robust in the case of backward congestion.

To remove the effect of reverse path congestion, we redefine the effective RTT

$$eRTT = RTT - d_{b,q} \quad (1)$$

$$d_{b,q} = d_b - min(d_b) \quad (2)$$

where *RTT* is a newly measured round trip time, *db,q* is the backward queueing delay, *db* is a measured backward delay, and *min*(*db*) is the minimum of all measured backward delays. Consequently, the *eRT T* indicates a round trip time when there is no backward path congestion.We compute the smoothed eRTT by using an exponential weighted moving average (EWMA), with a delay smoothing parameter of 1/8. This value is typically used for computing the smoothed RTT for TCP.

3.2 Congestion Control Based on Effective RTT

Congestion control is implemented in the internet using a *window flow control* algorithm. A source's *window* is the maximum number of unacknowledged packets that the source can inject into the network at any time. For example, if the window size is 1, then the source maintains a counter which has a maximum value of 1. The counter indicates the number of packets that it can send into the network. The counter's value is initially equal to the window size. When the source sends one packet into the network, the counter is reduced by 1. Thus, the counter in this example would become zero after each packet transmission and the source cannot send any more packets into the network till the counter hits 1 again. To increment the counter, the source waits for the destination to acknowledge that it has received the packet. This is accomplished by sending a small packet called the *ack* packet, from the destination back to the source. Upon receiving the ack, the counter is incremented by 1 and thus, the source can again send one more packet. We use the term *round-trip time (RTT)* to refer to the amount of time that elapses between the instant that the source transmits a packet and the instant at which it receives the acknowledgment for the packet. The RTT consists of three components: the propagation delay of the packet through the transmission medium (which is determined by the distance between the source and destination), the queueing delay at the routers in the network and the time taken to process a packet at the routers in the network. Typically, the processing time is negligible compared to the other two components With a window size of 1, since one packet is transmitted during every RTT, the source's data transmission rate is 1/*RTT* packets/sec. If the window is 2, the counter's value is initially set to 2. Thus, the source can send two back-to-back packets into the network. For each transmitted packet, the counter is decremented by 1. Thus, after the first two packet transmissions, the counter is decremented to zero. When one of the packets is acknowledged and the ack reaches the source, then the source increments the counter by 1 and can send one more packet into the network. Once the new packet is transmitted, the counter is again decremented back to zero. Thus, after each ack, one packet is sent, and then, the source has to wait for the next ack before it can send another packet. If one assumes that the processing speed of the link is very fast and that the processing times at the source and destination are negligible, then the source can transmit two packets during every RTT. Thus, the source's transmission rate is 2/*RTT* packets/sec. From the above argument, it should be clear that, if the window size is *W,* then the transmission rate can be approximated by*W/RTT* packets/sec. If the link capacity is *c* and the source's window size *W* is such that *W/RTT* < *c,* then the system will be stable. In other words, all transmitted packets will be eventually processed by the link and reach the intended destination. However, in a general network, the available capacity cannot be easily determined by a source. The network is shared by many sources which share the capacities at the various links in the network. Thus, each source has to adaptively estimate the value of the window size that can be supported by the network. Since previous research shows that HSTCP provides acceptable bandwidth scalability and friendliness [2], [5], we modified HSTCP's AIMD mechanisms as follows:

Additive Increase Algorithm:

The TCP Vegas estimates a proper amount of extra data to be kept in the network pipe (i.e. backlog) and controls the congestion window size accordingly. The amount is between the two thresholds α and β, as shown in the following:

$$\alpha \le N = (Expected - Actual) \times RTTmin \le \beta \quad (3)$$

where *Expected* is the current congestion window size divided by *RTTmin* (the minimum of all measured RTTs), and *Actual* represents the current congestion window size divided by the newly measured RTT. According to Little's Law, *N* represents the backlog at the bottleneck router queue. Thus, TCP Vegas tries to keep at least α packets, but no more than β packets queued in the network. For our scheme, first, to prevent throughput degradation from the reverse cross-traffic, we redefine *N* and *Actual* as follows:

$$Actual\_ = cwnd/eRTT \quad (4)$$

$$N\_ = (Expected - Actual\_) \times RTTmin = cwnd \times df,q/eRTT \quad (5)$$

where *df,q* is the forward queueing delay. Consequently, *N_* and *Actual_* represent the backlog and *Actual*, respectively, if there is no backward queueing delay. Since random noise in the RTT measurements (due to time resolution, OS interrupts, etc) cannot be avoidable in practice, FAST-like congestion control, which fully exploits delay to congestion control, seems unfeasible in most cases.

Multiplicative Decrease Algorithm:

After a packet loss, TCP halves the congestion window. If we size the router buffer to match the delay-bandwidth product, this mechanism ensures that the buffer does not underflow and goes empty. However, it is generally impractical; because there is no clear way to get average RTT information (even if it exists). Moreover, in high-speed networks large buffers are problematic for both technical as well as cost reasons. Enhanced Congestion Control Algorithm of TCP suggests a backoff scheme which makes a more informed decision by using minimum and maximum RTTs [11]. The rationale of the HTCP's backoff scheme is as follows: When congested, the total throughput through the link is given by

$$Throughput^{-} = \sum_{i=1}^{n} \frac{cwnd_i}{RTT_{max,i}} \quad (6)$$

where *n* is the number of flows and *RTTmax*,i is the maximum RTT experienced by the *i*'th source. After the backoff, the throughput is given by

$$Throughput^{+} = \sum_{i=1}^{n} \frac{(1-\beta_i) \times cwnd_i}{RTT_{min,i}} \quad (7)$$

To ensure the buffer is empty while preventing buffer underflow, HTCP sets $1 - \beta HTCP$ as *RTTmin/RT Tmax*. eHSTCP, in contrast to HTCP, uses:

$$1 - \beta_{eHSTCP} = \frac{RTT_{min}}{eRTT} \quad (8)$$

$$w \leftarrow w - w \times min(\beta_{eHSTCP}, \beta_{HSTCP}) \quad (9)$$

By inspecting the raw data from our simulation results, we found that the measured RTTs are frequently smaller than the maximum RTT when a packet loss occurs. The main reason behind this phenomenon is TCP burstiness. From the equation 9, eHSTCP reduces the congestion window by a smaller size than HSTCP. Reducing the congestion window less drastically improves utilization and throughput fluctuation but it hurts convergence speed and TCP friendliness since larger window flows give up their bandwidth slowly. To provide comparable TCP friendliness and bandwidth scalability of HSTCP at least while to avoid drastic decreasing congestion window.

4 Simulation Results and Discussion

In this section, we compare the simulated performance of eHSTCP with that of HSTCP, STCP, and BIC. Unless explicitly stated, the same amount of background traffic is used for all experimental runs. To reduce the phase effect and synchronized feedback, a significant amount of background traffic is

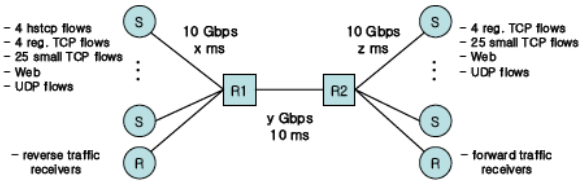

Fig. 1. The network topology for the simulation

Fig. 1. The network topology for the simulation eHSTCP: Enhanced Congestion Control Algorithm of TCP used in both directions, along with randomized RTTs and starting times. For background traffic, web traffic, 25 small TCP flows with a limited congestion window size under 64, and 4 long lived TCP flows are created in both directions for all simulations, unless otherwise specified.

The packet size is 1000 bytes. In our experiments, we use $N* = 10$ and the safety check phase $= 5 \times RTTmax$.

4.1 Utilization, Fairness, and Stability

In this experiment, RTT of all flows is around 40ms and the bottleneck bandwidth is 2.5Gbps. To evaluate bandwidth scalability, we measure link utilization and the average packet loss rate of the link between router R1 and R2. We also measure the fairness using Jain's fairness index among high-speed TCP flows. And the sample standard deviation normalized by the average throughput is used to evaluate stability. From Table 2, it can be seen that link utilization of eHSTCP is relatively comparable to that of STCP. Also, eHSTCP shows the best performance among all protocols under packet loss rate evaluation criterion. It is found that for HSTCP, BIC, and eHSTCP, the fairness index is approximately equal to 1 and STCP has some fairness issues. eHSTCP stays at the fully utilized region longer and proposed multiplicative decrease mechanism avoids unnecessarily drastic decreasing of congestion window. Therefore, we observe that eHSTCP shows the best stability.

4.2 RTT Fairness

In this experiment, two high speed flows with a different RTT are used. The RTT of flow 1 is 40ms, while the RTT of flow 2 is computed for 120ms and 240ms. The bottleneck bandwidth is 1Gbps. Table 3 depicts the

throughput ratio of the two high-speed flows. In Table 3, we see the bias against connections with long RTT. As predicted in [5], there is a serious fairness problem with flows of different RTTs for high speed TCP and STCP. High speedTCP and STCP tend to starve long RTT flows under high bandwidth environments, since short RTT flows quickly dominate the link bandwidth, starving out the other flows. eHSTCP's RTT fairness outperforms HSTCP, STCP and BIC.

4.3 TCP Friendliness

Transmission Control Protocol (TCP) has been widely used as a transport layer protocol with the explosive growth of the Internet. The Internet has operated stably with the TCP congestion control mechanism which avoids congestion collapse and achieves a reasonable throughput level. In the next generation Internet, new services such as multimedia transmissions and multicast applications will require different service criteria: a steady bandwidth guarantee, low delay, and a stable delay jitter. For new services, a transport layer protocol should guarantee the smooth bandwidth and a low delay jitter. However, TCP is not proper for new types of services because the throughput of the AIMD (Additive Increase Multiplicative Decrease)-based TCP congestion control algorithm oscillates rapidly even for slight losses. In addition, TCP has a retransmission mechanism for reliable transmission, and the TCP congestion window allows for the transmission of data packets in bursts. However, these TCP mechanisms increase throughput oscillation and delay jitter. Therefore, new services use UDP (User Datagram Protocol) as a transport layer protocol. But, since UDP has no congestion control mechanism, the increase of the UDP traffic for new services can cause network instability. TCP friendly protocols have been developed to overcome these defects of the current transport layer protocols for new services. TCP friendly congestion control protocols are designed to satisfy the following design issues. First, friendliness with the existing TCP should be provided. Since the Internet has been operated stably with the TCP congestion control mechanism to prevent congestion collapse, introducing a new protocol should not cause instability of the network. In addition, a TCP friendly protocol should not harm the conformant TCP throughput. Second, smoothness of the transmission rate should be satisfied. Rapid changes in the transmission rate deteriorate the service quality by increasing delay and delay jitter. Also, because the multimedia codec has the difficulty in changing data rate abruptly, the transmission method of TCP is not proper for multimedia communication. Third, responsiveness to a varying network state is required. Traffic multiplexing, router queue length, and routing update changes constantly the network state. A TCP friendly protocol should be able to adapt to various network states. Fourth, convergence to the fair share should be offered. When bandwidth is released, the protocol should converge to the fair share point as fast as possible. These properties conflict in some sense.

STCP achieves higher throughput for various scenarios but also, STCP shows the worst TCP friendliness followed by BIC and HSTCP in most cases. eHSTCP utilizes the link bandwidth as efficiently as HSTCP.

4.4 More Dynamic Scenario

In this scenario, we add 150 UDP flows with ON and OFF times drawn from a heavy-tailed distribution. The mean ON and OFF time is 1 second and the mean OFF time is also 1 second, with each source sending at 5Mbps during an ON time. Table 4 shows the percentage of the bandwidth shared by each flow type. And table 5 shows the average loss rate and the sample standard deviation normalized by the average throughput. BIC searches the equilibrium congestion window size by using loss history. In a dynamic scenario, loss history might be out of date, and thus unused bandwidth increases in BIC scenario. Note that the bandwidth scalability of eHSTCP is comparable to that of STCP and also, eHSTCP is the friendliest protocol of all the high-speed TCP protocols. To summarize, eHSTCP provides good TCP friendliness for all bandwidths while providing bandwidth scalability, which is comparable to STCP in high-speed environments.

5 Conclusion

In this paper, we propose a new variant of TCP for high-speed network which combines delay-based congestion control with loss-based congestion control. Although existing high-speed TCP schemes solve bandwidth scalability to some degree, there are still problems with fairness, friendliness, and stability. We define the effective RTT as the RTT that may be measured if there is no backward queueing delay along the path. Then, we refine HSTCP's AIMD mechanism. Since delay is error-prone, proposed additive increase algorithm uses effective RTT as binary feedback signal as to whether a network is full utilized. Proposed additive increase mechanism provides enhanced stability, reduced packet loss rate, and TCP friendliness. To guarantee the comparable TCP friendliness and scalability of HSTCP at least while to avoid drastic decreasing congestion window, proposed multiplicative decrease algorithm uses the effective RTT and deploys the safety check phase. We have shown through simulations that
the proposed scheme outperforms other high-speed TCPs in terms of fairness, friendliness, and stability, while utilizing a link bandwidth efficiently.

References

1. Katabi D., Handley M., and Rohrs C.: Internet Congestion Control for High Bandwidth-Delay Product Networks. In Proceedings of the ACM SIGCOMM, pp. 89-102, (2002).
2. Floyd S.: HighSpeed TCP for Large Congestion Windows. RFC3649, (2003).
3. Kelly T.: Scalable TCP: Improving Performance in Highspeed Wide Area Networks. ACM SIGCOMM Computer Communication Review, vol.33, pp. 83-91, (2003).
4. Jin C., Wei D. X. and Low S. H.: FAST TCP: motivation, architecture, algorithms, performance. In Proceedings of the IEEE Infocom, vol. 4, pp. 2490-2501, (2004).

5. Xu L., Harfoush K., and Rhee I.: Binary Increase Congestion Control for Fast, Long Distance Networks. In Proceedings of IEEE Infocom, vol. 4, pp. 2514-2524, (2004).
6. Brakmo L. and Peterson L.: TCP Vegas: End to End Congestion Avoidance on a Global Internet. IEEE Journal on Selected Areas in
7. Hengartner U., Bolliger J., and Gross T.: TCP Vegas Revisited. In Proceedings of IEEE Infocom, vol. 3, pp. 1546-1555, (2000).
8. Mo J., La R., Anantharam V., and Walrand J.: Analysis and Comparison of TCP Reno and Vegas. In Proceedings of IEEE Infocom, vol. 3, pp. 1546-1555, (2000).
9. . *F*eng W. and Vanichpun S.: Enabling compatibility between TCP Reno and TCP Vegas. In Proceedings of Symposium on Applications and the Internet, pp. 301-308, (2003).
10. Martin J., Nilsson A., and Rhee I.: Delay Based Congestion Avoidance for TCP. IEEE/ACM Transactions on Networking, vol. 11, no. 3, pp. 356-369, (2003).
11. Shorten R. and Leith D.: H-TCP: TCP for high-speed and long-distance networks. In Proceedings of the PFLDnet, (2004).
12. Choi Y., Lee K., and Cho Y.: Performance Evaluation of High-Speed TCP Protocols with Pacing. Lecture Notes in Computer Science, vol. 3332, pp. 322-329, (2004).