A Bottleneck and Target Bandwidth Estimates-Based Congestion Control Algorithm for High BDP Networks

Tuan-Anh Le[1], Choong Seon Hong[2]

Department of Computer Engineering, Kyung Hee University
1 Seocheon, Giheung, Yongin, Gyeonggi, 446-701, Korea
[1]letuanh@khu.ac.kr, [2]cshong@khu.ac.kr

***Abstract*— Two considerable problems for designing a new protocol in high bandwidth delay product (BDP) networks are bandwidth utilization efficiency and fairness to competing TCP flows. We introduce a novel congestion avoidance control algorithm for high BDP networks. By estimating the bottleneck and target bandwidth as the upper and lower bounds for increasing congestion window and decreasing congestion window. According to the current window size compared with those bounds, the congestion avoidance controller (CAC) adjusts the amount of appropriate congestion window increase in such a way that the sending rate stays around the point of the full link utilization. Decrease of the congestion window to a lower bound aids CAC to recover quickly after packet loss caused by either network congestion or random error. The proposal in simulation can achieve the bottleneck bandwidth utilization better than TCP-Illinois and HS-TCP do, and can improve fairness to competing flows mixed short RTT and long RTT.**

I. Introduction

In the today's Internet, a large number of high bandwidth links have been deploying, which in turn generates some technological challenges. Two of challenges are how to efficiently utilize large network bandwidth and fairly share network resources.

The standard TCP (TCP Reno) [1] employs a window growth algorithm to be an additive increase and multiplicative decrease mechanism. The data transmission control mechanism, in high BDP networks, requires TCP connections to maintain very large congestion window. However, it is slow for TCP Reno to obtain full utilization in the network resource if BDP of the TCP connection path is too large [2] because a congestion control algorithm of TCP Reno increases the congestion window (*cwnd*) by one packet in every round rip time (RTT) in congestion avoidance phase (CA) since no packet loss occurs, and reduces the *cwnd* by half if a packet loss is detected by means of three duplicate ACKs [1].

Specially, TCP connection paths include a wireless link. If packet losses occur by random errors of the wireless link, the sending rate is reduced blindly [5]. That is TCP performance to be degrading unreasonably.

In the recent research reports, these issues have been addressed and many solutions have proposed. These can be classified into four categories: loss-based approaches, delay-based approaches, mixed loss-delay-based approaches, and explicit congestion notification-based approaches. Protocols in loss-based approaches employ packet loss event as an indication of network congestion and try to modify the parameters of TCP Reno congestion control algorithm to be more aggressive in bandwidth utilization in high BDP networks. Such protocols consist of High Speed TCP (HS-TCP) [2], Scalable TCP [3], Binary Increase Congestion Control (BIC-TCP) [4], and CUBIC-TCP [5] (an improved BIC-TCP version). Delay-based approaches try to estimate RTT variations or queueing delay to detect an implicit incipient congestion indication, e.g., FAST TCP [6].

In mixed loss-delay-based approaches, proposals combine both delay-based and loss-based congestion indications. Compound TCP [7] maintains two different windows: the *cwnd* of TCP Reno is updated according to TCP Reno algorithm, the delay window (*dwnd*) increases rapidly when more bandwidth is available, but decreases when bottleneck queue is built-up. The actual congestion window is the sum of the *cwnd* and the *dwnd*; TCP-Illinois [8] employs packet loss as event in order to reduce *cwnd*, and senses queueing delay to determine an increase factor and multiplicative decrease factor during the CA.

EXplicit Control Protocol (XCP) [9] belongs to explicit congestion notification-based approaches. It needs intermediate router to signal to XCP about the degree congestion at the bottleneck. XCP maintains aggressiveness according to the available bandwidth and the feedback delay.

In this paper, we are interested in the mixed loss-delay-based approaches and introduce a congestion control algorithm, which estimates the bottleneck bandwidth and the target bandwidth (based on queueing delay information). It adjusts the amount of appropriate *cwnd* increase according to the current cwnd state compared with the target window in every RTT. When packet loss occurs, it sets the slow start threshold (*ssthresh*) and the cwnd equal to the estimated bottleneck window.

The rest of this paper is organized as follows: in Section II we address the shortcoming of the existing protocols, and list our objectives for design a good new protocol for high BDP networks. Section III articulates details of the congestion avoidance control algorithm of the proposal based on the bottleneck and target bandwidth estimates. The simulation results are aimed to evaluate TCP performace presented in Section IV. Finally we conclude in Section V.

II. Motivation

To reduce the *cwnd*, loss-based approaches employ packet loss after the full link utilization as congestion indication. After that, they are aggressive in the *cwnd* increase to achieve efficiency in high BDP network. However, this causes fluctuation in using queue at routers as well as unfairness in competing TCP flows with different RTTs. On the contrary, delay-based approaches try to adjust the *cwnd* around the full link utilization, which is based on queueing delay. However, they cannot share the same link with standard TCP [10], [11], and require the large buffer size at router for queueing flows [12].

To design a good new protocol for high BDP networks, in our opinion, following requirements should be considered:

Efficiency: The new protocol should efficiently utilize the large available bandwidth of high BDP networks with both only wired links and mixed wired-wireless links.

Fairness: In network with shared resources, the new protocol should fairly share the network bandwidth with the competing TCP flows in the same protocol. Additionally, under TCP flows with different RTTs, their average throughput can be achieved in inverse proportion to their RTTs. The flows with shorter RTT should not obtain shared bandwidth more compared with longer RTT flows.

Our proposal falls in with the same idea of TCP-Illinois and Compound TCP, which are based on packet loss event and queueing delay. However, in our proposal, queueing delay is employed to estimate the target window, which is a window at the full link utilization. According to the current *cwnd* state compared with the target window, the proposal appropriately adjusts the amount of increase for the *cwnd*. Furthermore, when packet loss occurs, the proposal sets the *ssthresh* and the *cwnd* equal to the bottleneck link window, which is based the estimated bandwidth.

III. The proposal

The proposed TCP's the congestion control mechanism is divided into two functional components called the estimator and the congestion avoidance controller (CAC). The estimator provides the bottleneck link bandwidth and target bandwidth for the CAC in order to control increase/decrease *cwnd*.

A. Bottleneck and Target Bandwidth Estimates

We take advantage of property of estimating the forward path not affected by ACK compression [12], [13]. The estimator computes the bottleneck link bandwidth by amount of data sent divided by inter-receiving time of that data packet via the ACK's timestamp [12], [13].

We define the bottleneck window ($W_{Bottleneck}$) based on the estimated bandwidth (*eBW*) [12], [13], as follows:

$$W_{Bottleneck} = \frac{eBW \cdot baseRTT}{Seg\_size}$$

where *baseRTT* is the mininum RTT observed, Seg_size is the length of the TCP segment.

To determine the target bandwidth in every RTT, the estimator employs the current window size (W), the maximum average RTT ($\overline{RTT}_{max}$) and the *baseRTT*. The total numbers of backlogged packets in the network are computed, as follows:

$$eQ = \frac{W}{\overline{RTT}}(\overline{RTT}_{max} - baseRTT)$$

where $\overline{RTT}_{max} - baseRTT$ is the maximum average total queueing delay in path of the flow and $W/\overline{RTT}$ is the estimated average current throughput, *eQ*) represents an estimate of the maximum total number of packets backlogged in the network.

The target window (W_{Target}) is the expected *cwnd* as fully utilizing the bottleneck bandwidth. The estimator computes

$$W_{Target} = W_{Bottleneck} + eQ\,.$$

The W_{Target} and $W_{Bottleneck}$ determine the upper and lower bounds of bandwidth for a TCP flow. They are reflecting the current network condition by updating in every RTT.

B. The Congestion Avoidance Control Algorithm

Given $W_{Bottleneck}$ and W_{Target} from the estimator, CAC increases the *cwnd* by $\alpha(.)$ packets in every RTT, as follows:

$$cwnd(t+1) = cwnd(t) + \frac{\alpha(cwnd(t))}{cwnd(t)}. \qquad (1)$$

Function $\alpha(.)$ is exponential to be growth of the *cwnd*, according to:

$$\alpha(w) = \begin{cases} \alpha_{min} + \ell . e^{-\lambda(w - W_S)} & , if W_S \le w \le W_{Target}, \\ \alpha_{max} & , otherwise \end{cases}$$

where $\ell = \alpha_{max} - \alpha_{min}$, $W_S = W_{Bottleneck} + 0.2 \cdot (W_{Target} - W_{Bottleneck})$ is the window size at which the CAC switches none-linear increase mode for the *cwnd*. Parameters of α_{min}, α_{max}, and λ determine aggressiveness of the protocol as in Fig.1(a). In this paper, we set α_{min} to be 1.0, α_{max} to be 16.0. λ is set to 7.25/($W_{Target} - W_{Bottleneck}$)(such λ value means that an *cwnd* is at 2/3 times the distance from $W_{Bottleneck}$ to W_{Target}, where it meets a value of the sum of min and 20% of the distance from α_{min} to α_{max}).

In CA phase, the *cwnd* is increased the amount of increase according to the current *cwnd* state compared with both the target window and the bottleneck window in every RTT. Fig.1(b) shows curve of the *cwnd* growth function of the proposal according to the function $\alpha(.)$ as in Fig.1(a).

We separate the *cwnd* growth as in (1) into two increment phases as follows. The first increment phase is determined if the current *cwnd* is in between $W_{Bottleneck}$ and W_S (It means that the current *cwnd* is far from the target window). The CAC interprets that more bandwidth are available. Then, the *cwnd* is increased by the largest value (i.e., α_{max}, called the

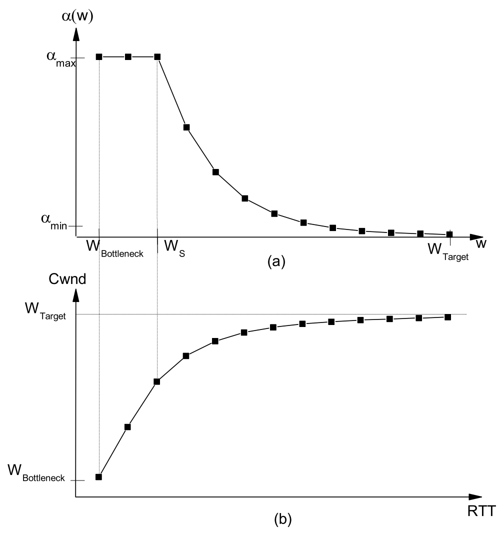


Fig. 1. (a) Curve of function $\alpha(w)$, and (b) curve of *cwnd* growth function.

linear increment) provided from the function $\alpha(.)$. The second increment phase is determined if the current *cwnd* is close the target window. The CAC interprets that the network is becoming the full link utilization. Then, the *cwnd* is increased by a small value provided from the function $\alpha(.)$. The amount of *cwnd* increase is smaller; it is useful to keep the *cwnd* to stay around the point of the full link utilization.

After replacing the increment function of the *cwnd* in TCP Reno protocol during the CA phase as in above, we need to modify its congestion avoidance control algorithm in two following packet loss events.

Whenever the packet loss is detected through receiving triple duplicate ACKs, the CAC updates the *ssthresh* and the *cwnd*, as follows, and starts a new CA phase:

$$\begin{aligned} ssthresh &\leftarrow max(W_{Bottleneck}, 2) \\ cwnd &\leftarrow ssthresh. \end{aligned} \tag{2}$$

The equation (2) interprets that the CAC reduces the data transmission rate equal to the rate limitation of the bottleneck link as router's queue underflows. This is useful to the CAC to begin at a proper point (i.e. the bottleneck link capacity) for *cwnd* recovery.

If the CAC is triggered by a retransmission timeout event due to the heavy network congestion or very high bit-error rate of wireless link, the CAC sets *ssthresh* to $W_{Bottleneck}$ and then sets *cwnd* to one for restarting the slow start phase, and resets $\overline{RTT}_{max}$ to zero as the following pseudo code:

$$\begin{aligned} ssthresh &\leftarrow max(W_{Bottleneck}, 2) \\ cwnd &\leftarrow 1 \\ \overline{RTT}_{max} &\leftarrow 0. \end{aligned}$$

IV. Performance Evaluations

In this section we evaluate the proposed TCP compared with HS-TCP, and TCP-Illinois. The evaluations are performed by the NS-2 simulator [14]. The proposed protocol is implemented in TCP-Linux [15]. A dumbbell scenario is employed in order to obtain and compare the fundamental features of the protocols such as congestion window evolution, utilization, and fairness.

Such scenario configuration is shown in Fig. 2 with the one way link delay of 80ms, and the bottleneck link speed of 200Mbps (i.e. one way BDP capacity of 1333 packets/s with packet size 1500-byte), the routers' buffer size of 400 packets (with the drop tail queue). The TCP packet size is 1500-byte. The simulation was run for 500 seconds.

A. The Congestion Window Evolution

We visualized the congestion window evolution of a single flow as in Fig. 3. It is observed that the proposed TCP and TCP-Illinois can stay around buffer range allocated at the bottleneck buffer. However, the proposed TCP does not reduce the *cwnd* lower than the bottleneck link capacity, and its increment is more aggressive in than TCP-Illinois' increment. On the contrary, HS-TCP's *cwnd* is fluctuated significantly in acquiring and releasing bandwidth.

B. Utilization

Utilization of a TCP protocol is determined by how to use efficiently the bottleneck link bandwidth in term of TCP googput under both high BDP and random packet error environments.

In first environment, we consider the impact of the bottleneck bandwidth of link on utilization along bottleneck bandwidth variants from 100Mbps to 500Mbps.

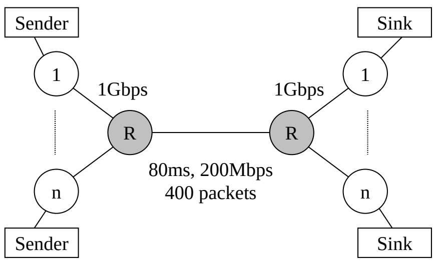


Fig. 2. The dumbbell simulation scenario.

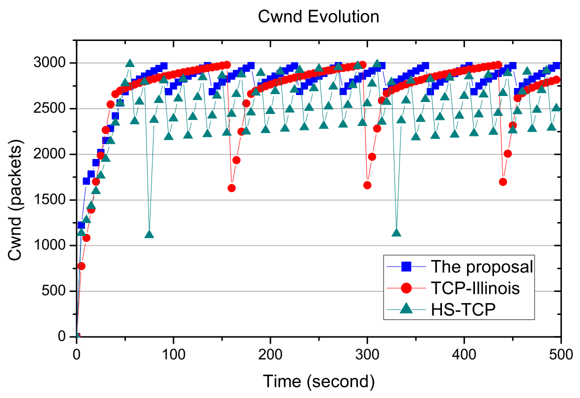


Fig. 3. The congestion window evolution.

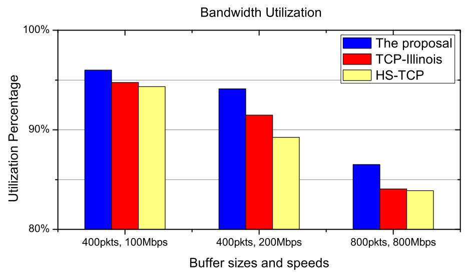


Fig. 4. The bandwidth utilization efficiency.

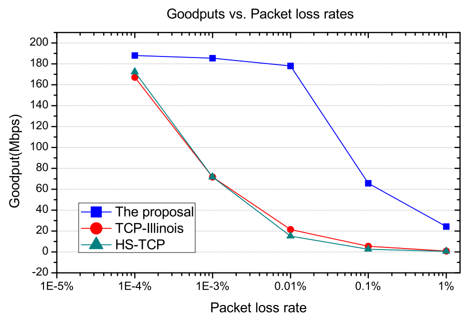


Fig. 5. The goodput under various random packet loss rates.

Fig. 4 shows that utilization efficiency of the protocols are over 80% of the bottleneck bandwidth, but the proposed TCP is higher than TCP Illinois and HS-TCP in all three simulations. However, when the bottleneck bandwidth is given larger, more bandwidth may be under-utilized. This is due to setting the scalable parameters of these propocols.

In second environment, we evaluate goodputs of all three protocols under random packet loss rate varying from 0.0001% to 1%. In Fig. 5, for any random packet loss rate, the goodput of the proposed TCP is better than other TCP protocols. Practically, at 0.01% random packet loss rate, the proposed TCP outperforms TCP Illinois and HS-TCP by 7.3 times and 10.7 times, respectively. This is interpreted that the proposed TCP can quickly recover from the random packet loss, which is not related to the queueing drop.

C. Fairness

We consider fairness as fairly sharing the network bandwidth with the competing TCP flows in the same protocol. To measure of fairness, we employed Jain's fairness index [16], as follows:

$$F_{index} = \frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)}, \frac{1}{n} \leq F_{index} \leq 1.0$$

where x_i is the throughput of the i'th TCP flow, and n is the number of TCP flows.

The numbers of TCP flows in our simulations are varied from 4 flows to 16 flows. Fig. 6 shows that the proposed TCP can achieve quite higher fairness index than TCP-Illinois and HS-TCP. The main reason why TCP-Illinois gets lowest fairness index is read off via plotting *cwnd* evolution of 4 flows of the protocols in Fig. 7.

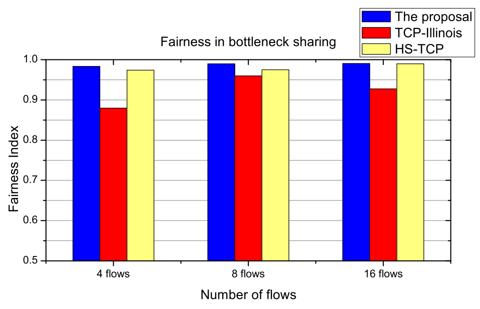


Fig. 6. The fairness index in bottleneck sharing.

TABLE I

THE AVERAGE GOODPUT RATIO UNDER DIFFERENT RTTS.

Inverse RTT ratio	**1.5**	**2**	**3**
Proposed TCP	1.68	2.08	2.80
TCP-Illinois	1.96	3.6	4.1
HS-TCP	7.62	40.64	247.24

Figs. 7(a) and (c) show that the flows' *cwnds* of proposed TCP and HS-TCP can converge to around fairly shared bandwidth at 750 packets. On the contrary, the flows' cwnd evolutions of TCP Illinois in Fig. 7(b) are significantly different from each other through simulation time.

We next evaluate RTT unfairness to competing TCP flows with different RTTs, where the *cwnd* of a short RTT TCP flow increases faster than that of long RTT TCP flow so that the average throughput ratio between the short RTT flow and long RTT flow is inversely proportional to their RTT ratio [4].

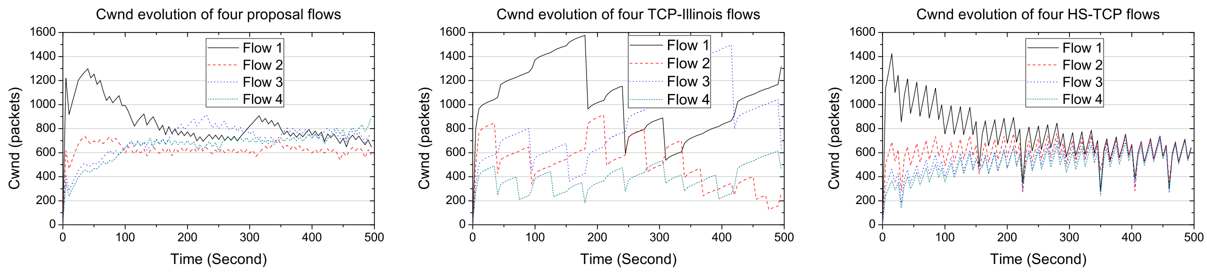
In this simulation, a short RTT flow is competing with a long RTT one. The short RTT flow has RTT of 80ms. The RTT of the long RTT flow is varied among 120ms, 160ms, and 240ms. Table I shows that HS-TCP suffers from highest unfairness as the inverse RTT ratio increase. The proposed TCP is fairer to competing TCP flow with different RTTs.

V. CONCLUSIONS

In this paper, we present a novel congestion avoidance control algorithm for high BDP networks. The proposal can satisfy two essential requirements for designing a new protocol in high BDP networks. By estimating the bottleneck and target bandwidth based on queueing delay and bottleneck link rate, the proposal increases cwnd in logarithmic form in such a way that the sending rate stays around the point of the full link utilization. Decrease of the cwnd to bottleneck capacity aids CAC to recover quickly after packet loss caused by either network congestion or random error.

The simulation results show that the proposal can achieve the bottleneck bandwidth utilization better than TCP-Illinois and HS-TCP, and can improve fairness to competing flows mixed short RTT and long RTT.

However, executing the increase function is slow in implementation because of natural exponential function. We can find another equivalent exponential function or implement a fast look-up method to compute that function as in [17].



(a) (b) (c)

Fig. 7. The *cwnd* evolution of four flows of (a) our proposal, (b) TCP-Illinois, and (c) HS-TCP.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT [2009-S-014-01, On the Development of Sensing based Emotive Service Mobile Handheld Devices].

Dr. CS Hong is corresponding author.

REFERENCES

[1] V. Jacobson and M. J. Karels, “Congestion avoidance and control,” in *SigCOMM 88*, 1988.

[2] S. Floyd, “Highspeed TCP for large congestion windows,” RFC 3649, Dec. 2003.

[3] T. Kelly, “Scalable TCP: Improving performace in highspeed wide area networks,” *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 83–91, 2002.

[4] L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control (BIC) for fast long-distance networks,” in *Proc. IEEE INFOCOM’04*, Mar. 2004.

[5] S. Ha, I. Rhee, and L. Xu, “CUBIC: A new TCP-friendly high-speed TCP variant,” in *ACM SIGOPS Operating System Review*, July 2008.

[6] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, “FAST TCP: motivation, architecture, algorithms, performance,” in *Proc. IEEE INFOCOM’04*, Mar. 2004.

[7] K. Tan, J. Song, Q. Zhang, and M. Sridharan, “A Compound TCP approach for high-speed and long distance networks,” Microsoft Research TechReport, Tech. Rep., 2005. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=70189

[8] S. Liu, T. Basar, and R. Srikant, “TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks,” in *Proc. VALUETOOLS*, Oct. 2006.

[9] D. Katabi, M. H, L. C. Rohrs, and M. lcs Icsi Tellabs, “Congestion control for high bandwidth-delay product networks,” in *Proc. ACM SIGCOMM*, 2002.

[10] J. Mo, R. J. La, V. Anantharam, and J. Walrand, “Analysis and comparison of TCP Reno and Vegas,” in *Proc. IEEE INFOCOM’99*, Mar. 1999.

[11] A. Tang, J. Wang, S. Low, and M. Chiang, “Equilibrium of heterogeneous congestion control protocols,” in *Proc. IEEE INFOCOM’05*, Mar. 2005.

[12] T. A. Le and C. S. Hong, “TCP BaLDE for improving TCP performance over heterogeneous networks,” *IEICE Transactions on Communications*, vol. E89-B, No.4, pp. 1127–1135, Apr. 2006.

[13] ——, “Stable accurate rapid bandwidth estimate for improving TCP over wireless networks,” in *Lecture Notes in Computer Science 3421 (ICN 2005)*, Apr. 2005.

[14] (2009) Ns-2 network simulator, version 2.34. [Online]. Available: http://www.isi.edu/nsnam/ns/

[15] D. X. Wei and P. Cao. (2006) A linux tcp implementation for ns2. [Online]. Available: http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux-2.29-linux-2.6.16/

[16] R. Jain, D. Chiu, and W. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,” DEC Research, Tech. Rep. TR-301, September 1984. [Online]. Available: http://www.cse.wustl.edu/ jain/papers/fairness.htm

[17] C. Baumann. (2004) A simple and fast look-up table method to compute the exp(x) and ln(x) functions. [Online]. Available: http://www.convict.lu/Jeunes/ultimate_stuff/exp_ln_2.htm