CUBIC: A New TCP-Friendly High-Speed TCP Variant

Injong Rhee, and Lisong Xu

***Abstract*— This paper presents a new TCP variant, called CUBIC, for high-speed network environments. CUBIC is an enhanced version of BIC: it simplifies the BIC window control and improves its TCP-friendliness and RTT-fairness. The window growth function of CUBIC is governed by a cubic function in terms of the elapsed time since the last loss event. Our experience indicates that the cubic function provides a good stability and scalability. Furthermore, the real-time nature of the protocol keeps the window growth rate independent of RTT, which keeps the protocol TCP friendly under both short and long RTT paths.**

***Index Terms*—Congestion Control, High-Speed TCP, TCP Friendliness**

I. Introduction

In the past few years, we have witnessed a surge of TCP variants [1] that address the under-utilization problem most notably due to the slow growth of TCP congestion window (e.g., FAST [2], HSTCP [3], STCP [4], HTCP [5], SQRT [6], Westwood [7], and BIC [8]) in a large bandwidth-delay product (BDP) network. Most of these protocols deal with modifying the window growth function of TCP in a more scalable fashion. While scaling the window growth rate to match large BDP is rather straightforward, tackling the fairness issues of new protocols has remained as a major challenge. These fairness issues include friendliness to existing TCP traffic (TCP friendliness), and fair bandwidth sharing with other competing high-speed flows running with same or different round-trip delays (Inter/intra protocol fairness, and RTT fairness).

TCP-friendliness defines whether a protocol is being fair to TCP, and it is critical to the safety of the protocol. When a protocol is used, we need to make sure that its use does not unfairly affect the most common network flows (namely TCP). Many different definitions of this property are found in the literature. The most commonly cited one is by [3]: under high loss rate regions where TCP is well-behaving, the protocol must behave like TCP, and under low loss rate regions where TCP has a low utilization problem, it can use more bandwidth than TCP. Most high-speed TCP variant protocols achieve TCP friendliness by having some form of "TCP modes" during which they behave in the same way as TCP. HSTCP, STCP, and BIC enter their TCP modes when the window size is less than some small cutoff constant (typically around 30 packets).

However, our observation is that the regime where TCP performs well (called "TCP region") should be defined by the congestion epoch time (not by the window size) – the real-time period between two consecutive loss events. Although BDP implies the network capacity by packet count (or window size), packet count is not adequate to characterize TCP performance because the growth rate of TCP depends on RTT. For instance, TCP can grow to 30 packets in 30 ms with 1 ms RTT while in 3 seconds with 100 ms RTT. Even in a network with the bottleneck bandwidth of 1 Gbps, TCP can grow the window up to the full utilization of the network within 100 ms, if RTT is 1ms. A protocol like HSTCP, STCP, and BIC will operate mostly in a "scalable" mode in this network, possibly consuming bandwidth unfairly over other competing TCP traffic. Note that the scalability problem of TCP is often defined by real-time (recall quotes from many folks saying "TCP takes more than one hour to reach the full utilization of so-and-so networks"). Thus, it seems more appropriate (or better) to define the TCP region by real-time. While the exact time period of the TCP region is as debatable as the window size of the TCP region of the above high-speed protocols, we can still define a protocol that is amenable to the real-time definition of the TCP region.

In this paper, we propose yet another variant of TCP, called CUBIC, that enhances the fairness properties of BIC while retaining its scalability and stability. The main feature of CUBIC is that its window growth function is defined in real-time so that its growth will be independent of RTT. Our work was partially inspired by HTCP [5], whose window growth function is also based on real time. The congestion epoch period of CUBIC is determined by the packet loss rate alone. As TCP's throughput is defined by the packet loss rate as well as RTT, the throughput of CUBIC is defined by only the packet loss rate. Thus, when the loss rate is high and/or RTT is short, CUBIC can operate in a TCP mode. Moreover, since the growth function is independent of RTT, its RTT fairness is guaranteed as different RTT flows will still grow their windows at the same rate.

II. CUBIC – A New TCP Variant

CUBIC is an enhanced version of BIC. It simplifies the BIC window control and improves its TCP-friendliness and RTT-fairness.

Injong Rhee is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7534 USA (corresponding author, phone: 919-515-3305; fax: 919-515-7925; e-mail: rhee@csc.ncsu.edu).

Lisong Xu is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115 USA (e-mail: xu@cse.unl.edu).

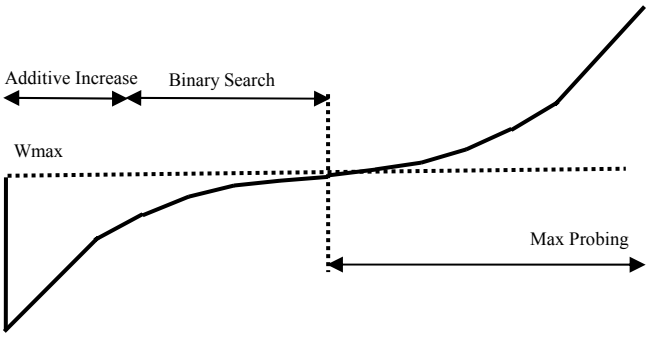

Fig. 1: The Window Growth Function of BIC

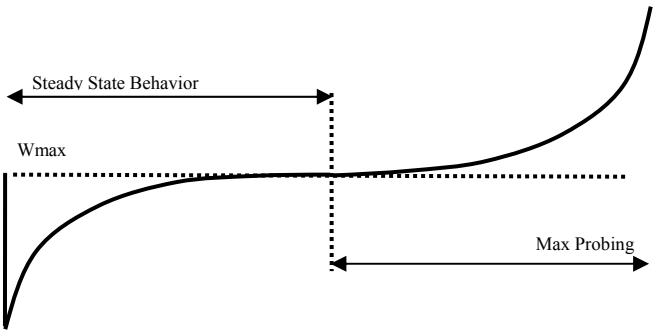

Fig. 2: The Window Growth Function of CUBIC

A. BIC Window Growth Function

Before delving into CUBIC, let us examine the features of BIC. The main feature of BIC is its unique window growth function.

Fig. 1 shows the growth function of BIC. When it gets a packet loss event, BIC reduces its window by a multiplicative factor β. The window size just before the reduction is set to the maximum W_{max} and the window size just after the reduction is set to the minimum W_{min}. Then, BIC performs a binary search using these two parameters – by jumping to the "midpoint" between W_{max} and W_{min}. Since packet losses have occurred at W_{max}, the window size that the network can currently handle without loss must be somewhere between these two numbers.

However, jumping to the midpoint could be too much increase within one RTT, so if the distance between the midpoint and the current minimum is larger than a fixed constant, called S_{max}, BIC increments the current window size by S_{max} (linear increase). If BIC does not get packet losses at the updated window size, that window size becomes the new minimum. If it gets a packet loss, that window size becomes the new maximum. This process continues until the window increment is less than some small constant called S_{min} at which point, the window is set to the current maximum. So the growing function after a window reduction will be most likely to be a linear one followed by a logarithmic one (marked as "additive increase" and "binary search" respectively in Fig. 1).

If the window grows past the maximum, the equilibrium window size must be larger than the current maximum and a new maximum must be found. BIC enters a new phase called "max probing." Max probing uses a window growth function exactly symmetric to those used in additive increase and binary search – only in a different order: it uses the inverse of binary search (which is logarithmic; its reciprocal will be exponential) and then additive increase. Fig. 1 shows the growth function during max probing. During max probing, the window grows slowly initially to find the new maximum nearby, and after some time of slow growth, if it does not find the new maximum (i.e., packet losses), then it guesses the new maximum is further away so it switches to a faster increase by switching to additive increase where the window size is incremented by a large fixed increment.

The good performance of BIC comes from the slow increase around W_{max} and linear increase during additive increase and max probing.

B. CUBIC Window Growth Function

Although BIC achieves pretty good scalability, fairness, and stability during the current high speed environments, the BIC's growth function can still be too aggressive for TCP, especially under short RTT or low speed networks. Furthermore, the several different phases of window control add a lot of complexity in analyzing the protocol. We have been searching for a new window growth function that while retaining most of strengths of BIC (especially, its stability and scalability), simplifies the window control and enhances its TCP friendliness.

In this paper, we introduce a new high-speed TCP variant: CUBIC. As the name of the new protocol represents, the window growth function of CUBIC is a cubic function, whose shape is very similar to the growth function of BIC. CUBIC is designed to simplify and enhance the window control of BIC. More specifically, the congestion window of CUBIC is determined by the following function:

$$W_{cubic} = C(t-K)^3 + W_{max} \quad (1)$$

where C is a scaling factor, t is the elapsed time from the last window reduction, W_{max} is the window size just before the last window reduction, and $K = \sqrt[3]{W_{max}\beta/C}$, where β is a constant multiplication decrease factor applied for window reduction at the time of loss event (i.e., the window reduces to βW_{max} at the time of the last reduction).

Fig. 2 shows the growth function of CUBIC with the origin at W_{max}. The window grows very fast upon a window reduction, but as it gets closer to W_{max}, it slows down its growth. Around W_{max}, the window increment becomes almost zero. Above that, CUBIC starts probing for more bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from W_{max}. This slow growth around W_{max} enhances the stability of the protocol, and increases the utilization of the network while the fast growth away from W_{max} ensures the scalability of the protocol.

The cubic function ensures the intra-protocol fairness among the competing flows of the same protocol. To see this, suppose that two flows are competing on the same end-to-end path. The

two flows converge to a fair share since they drop by the same multiplicative factor β – so a flow with larger W_{max} will reduce more, and the growth function allows the flow with larger W_{max} will increase more slowly – K is larger as W_{max} is larger. Thus, the two flows eventually converge to the same window size.

The function also offers a good RTT fairness property because the window growth rate is dominated by t, the elapsed time. This ensures linear RTT fairness since any competing flows with different RTT will have the same t after a synchronized packet loss (note that TCP and BIC offer square RTT fairness in terms of throughput ratio).

To further enhance the fairness and stability, we clamp the window increment to be no more than S_{max} per second. This feature keeps the window to grow linearly when it is far away from W_{max}, making the growth function very much in line with BIC's as BIC increases the window additively when the window increment per RTT becomes larger than some constant. The difference is that we ensure this linear increase of the CUBIC window to be real-time dependent— when under short RTTs, the linear increment per RTT is smaller although stays constant in real time.

C. New TCP Mode

The real-time increase of the window enormously enhances the TCP friendliness of the protocol. Note that the window growth function of other RTT dependent protocols (TCP being a good example), grows proportionally faster (in real time) in shorter RTT networks whereas CUBIC will grow independently of RTT. Since TCP gets more aggressively while CUBIC being unchanged, short RTT will make CUBIC more friendly to TCP. In short RTT networks, CUBIC's window growth is slower than TCP.

In order to keep the growth rate the same as TCP, we emulate the TCP window adjustment algorithm after a packet loss event. Since CUBIC reduces its window by a factor of β after a loss event, the TCP-fair additive increment would be $3\left((1-\beta)/(1+\beta)\right)$ per RTT. This is because the average sending rate of an Additive Increase Multiplicative Decrease protocol (AIMD) is

$$\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\frac{1}{p}} \tag{2}$$

where α is the additive window increment, and p is the loss rate. For TCP, α=1 and β=1/2, so the average sending rate of TCP is.

$$\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}} \tag{3}$$

To achieve the same average sending rate as TCP with an arbitrary β, we need α equal to $3\left((1-\beta)/(1+\beta)\right)$. If we set β to 0.8, the additive increase factor is 0.5. Given this growth rate per RTT, the window size of emulated TCP at time t (after the last epoch) is

$$W_{tcp} = W_{\max}\beta + 3\frac{1-\beta}{1+\beta}\frac{t}{RTT} \tag{4}$$

If W_{tcp} is larger than W_{cubic} (in eqn. 1), then we set the window size to W_{tcp}. Otherwise, W_{cubic} is the current congestion window size. Our analysis, proof omitted, shows that if the congestion epoch duration is less than $1/\sqrt{C \cdot RTT}$, or if the packet loss rate is larger than $0.36 \cdot C \cdot RTT^3$, then CUBIC is TCP-friendly. With C=0.4 and RTT=100ms, when the packet loss rate is larger than 0.000144, CUBIC is TCP friendly. Compared to HSTCP where it is TCP friendly when the loss rate is larger than 0.001, CUBIC has a larger area of the TCP friendly region. Further, when the RTT is very small, CUBIC is much more TCP friendly than HSTCP regardless of loss rates.

D. CUBIC in Action

Fig. 3 shows the window curve of CUBIC over the running time. This graph is obtained by running an NS simulation experiment on a dumbbell network configuration with significant background traffic in both directions to remove the phase effect. The bottleneck capacity is 500Mbps and the RTT is set to 100ms. Drop Tail routers are used. There are two CUBIC flows and two TCP flows, and all flows have the same RTT and bottleneck. Note that the curves have plateaus around W_{max} which is the window size at the time of the last packet loss event.

Fig. 4 shows the same NS simulation but for a longer time. We observe that the two flows of CUBIC converge to a fair share nicely around 220 seconds. Their CWND curves are very smooth and do not cause much disturbance to competing TCP flows. In this experiment, the total network utilization is around 98%: the two CUBIC flows take about 78% of the total bandwidth, the two TCP flows take 11%, and the background flows take up around 9%.

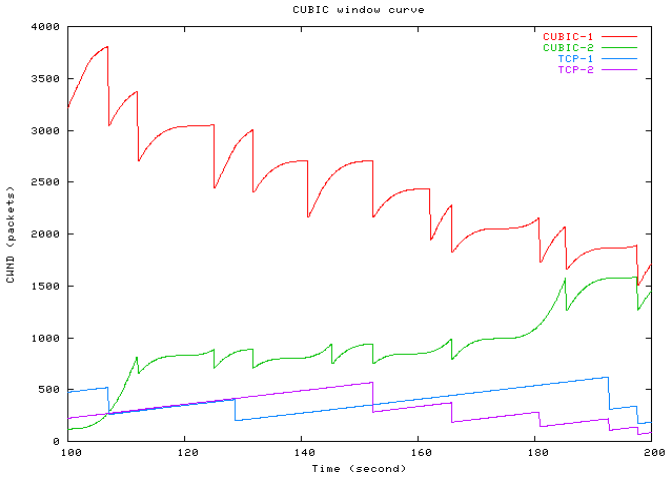

Fig. 3: CUBIC window curves (NS simulation in a network with 500Mbps and 100ms RTT), C = 0.4, β = 0.8.

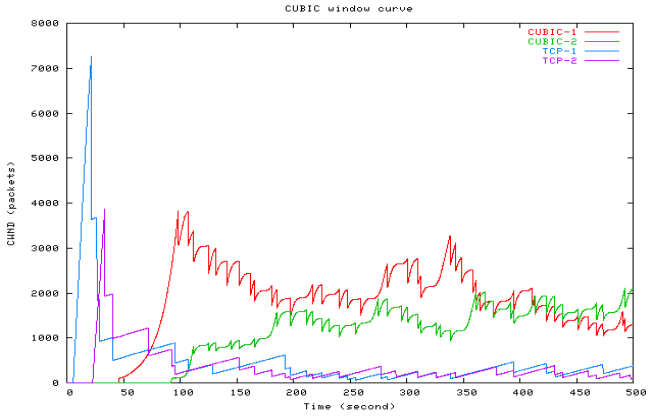

Fig. 4: CUBIC window curves with competing flows (NS simulation in a network with 500Mbps and 100ms RTT), $C = 0.4$, $\beta = 0.8$.

III. Performance Evaluation

In this section, we present some performance results regarding the TCP friendliness and stability of CUBIC and other high-speed TCP variants. For CUBIC, we set β to 0.8, C to 0.4, and S_{max} to 160. We use NS-2 for simulation. The network topology is dumbbell. For each simulation run, we run four flows of a high-speed protocol and four flows of regular long-term TCP SACK over the same end-to-end paths for the entire duration of the simulation; their starting times and RTTs are slightly varied to reduce the phase effect. About 10% of background traffic is added in both forward and backward directions of the dumbbell setup. For all the experiments unless notes explicitly, the buffer size of Drop Tail routers is set to 100% of BDP.

Experiment 1: TCP Friendliness in Short-RTT Networks (Simulation script available in the BIC web site):

We test five high speed TCP variants: CUBIC, BIC, HSTCP, Scalable TCP, and HTCP. We set RTT of the flows to be around 10 ms and vary the bottleneck bandwidth from 20 Mbps to 1 Gbps. Fig. 5 shows the throughput ratio of the long-term TCP flows over the high-speed flows (or TCP friendly ratio) measured from these runs.

The surprising result is that BIC and STCP even show worse TCP friendliness over 20Mbps than over 100Mbps. However, we are still not sure the exact reason for this result. Over 100 Mbps, all the high speed protocols show reasonable friendliness to TCP. As the bottleneck bandwidth increases from 100Mbps to 1Gbps, the ratios for BIC, HSTCP and STCP drop dramatically indicating unfair use of bandwidth with respect to TCP. Under all these environments, regular TCP can still use the full bandwidth. Scalable TCP shows the worst TCP friendliness in these tests followed by BIC and HSTCP. CUBIC and HTCP consistently give good TCP friendliness.

Experiment 2: TCP Friendliness in Long-RTT Networks (Simulation script available in the BIC web site)

Although the TCP mode improves the TCP friendliness of the protocol, it does so mostly for short RTT situations. When the BDP is very large with long RTT, the aggressiveness of the window growth function (more specifically, the congestion epoch length) has more decisive effect on the TCP friendliness. As the epoch gets longer, it gives more time for TCP flows to grow their windows.

An important feature of BIC and CUBIC is that it keeps the epoch fairly long without losing scalability and network utilization. Generally, in AIMD, a longer congestion epoch means slower increase (or a smaller additive factor). However, this would reduce the scalability of the protocol, and also the network would be underutilized for a long time until the window becomes fully open (Note that it is true only if the multiplicative decrease factor is large; but we cannot keep the multiplicative factor too small since that implies much slower convergence to the equilibrium). Unlike AIMD, CUBIC increases the window to (or its vicinity of) W_{max} very quickly and then holds the window there for a long time. This keeps the scalability of the protocol high, while keeping the epoch long and utilization high. This feature is unique both in BIC and CUBIC.

In this experiment, we vary the bottleneck bandwidth from 20Mbps to 1Gbps, and set RTT to 100ms. Fig. 6 shows the throughput ratio of long-term TCP over high-speed TCP variants. Over 20 Mbps, all the high speed protocols show reasonable friendliness to TCP. As the bandwidth gets larger than 20 Mbps, the ratio drops quite rapidly. Overall, CUBIC shows a better friendly ratio than the other protocols.

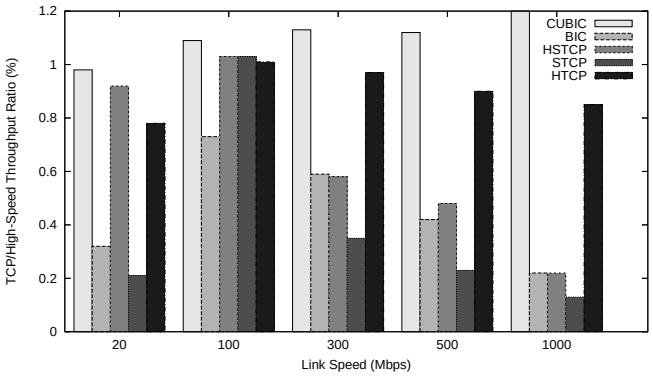

Fig. 5: TCP-Friendly Ratio in Short-RTT Networks

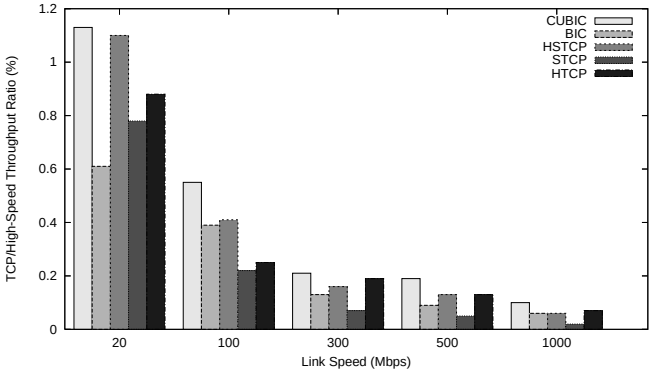

Fig. 6: TCP-Friendly Ratio in Long-RTT Networks

Experiment 3: Stability (Simulation script available in the

BIC web site)

We run four flows of a high-speed TCP variant over a long-RTT network path (~220ms) and four flows of long-term TCP-SACK flows over a short-RTT path (~20ms). These two paths share a bottleneck link of 2.5Gbps. In this experiment, to see how stable different protocols become as the buffer space of the bottleneck router varied, we vary the buffer space of the bottleneck router from 200% to 20% of the BDP of the bottleneck. The background TCP traffic is added to the bottleneck link. Fig. 7 illustrates our simulation setup (slightly modified for clarity). The actual simulation setup can be found in the script above.

Below, we show the throughput graphs from experiments with 20% buffer in Fig. 8, and with 200% buffer in Fig. 9. By inspecting the raw data, we can tell that STCP and HTCP have some stability issues (this needs to be confirmed with the original authors of HTCP). The high oscillation occurs over various time scales.

There is no well-defined metric of stability. Existing literature on congestion control often uses the smoothness in transmission rate variations (or smaller oscillations) to mean stability. Control theory defines it somewhat differently: a stable protocol eventually converges to equilibrium (not necessarily a fair bandwidth share) regardless of the current state of the protocol. These two notions are somewhat connected since a protocol would have a very small oscillation once it converges to equilibrium, and they are not necessarily the same. Often the coefficients of variation (CoV) of transmission rates are used to depict stability as some artificial perturbations to the traffic are added to the network. However, since network environments constantly change, the transmission rate of a protocol always fluctuates at a short-term scale. Then, what would be an appropriate time scale to determine its stability? We are currently investigating techniques to measure the average fairness index (by Jain) at various time scales and compare those of various protocols. For a less satisfactory measure, we plotted the coefficients of variance (CoV) of throughput. This metric is also used in [9,10,11]. The results with 20% buffer are shown in Fig 10, and the results with 200% buffer are shown in Fig 11. We observe that CUBIC shows a good stability.

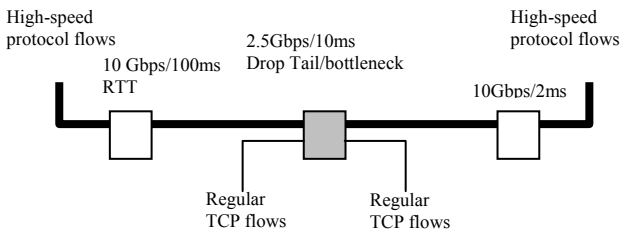

Fig. 7: Simulation setup for stability test.

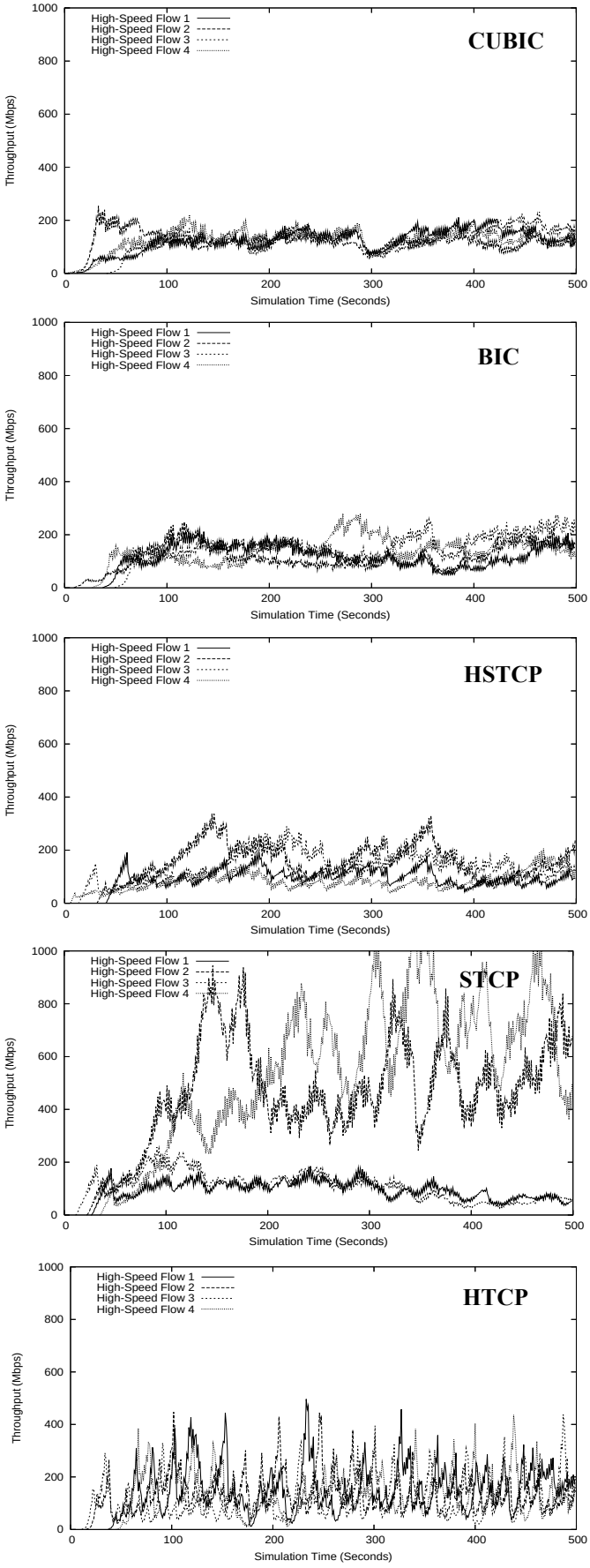

Fig. 8: Throughput various protocols in stability test with 20% buffer

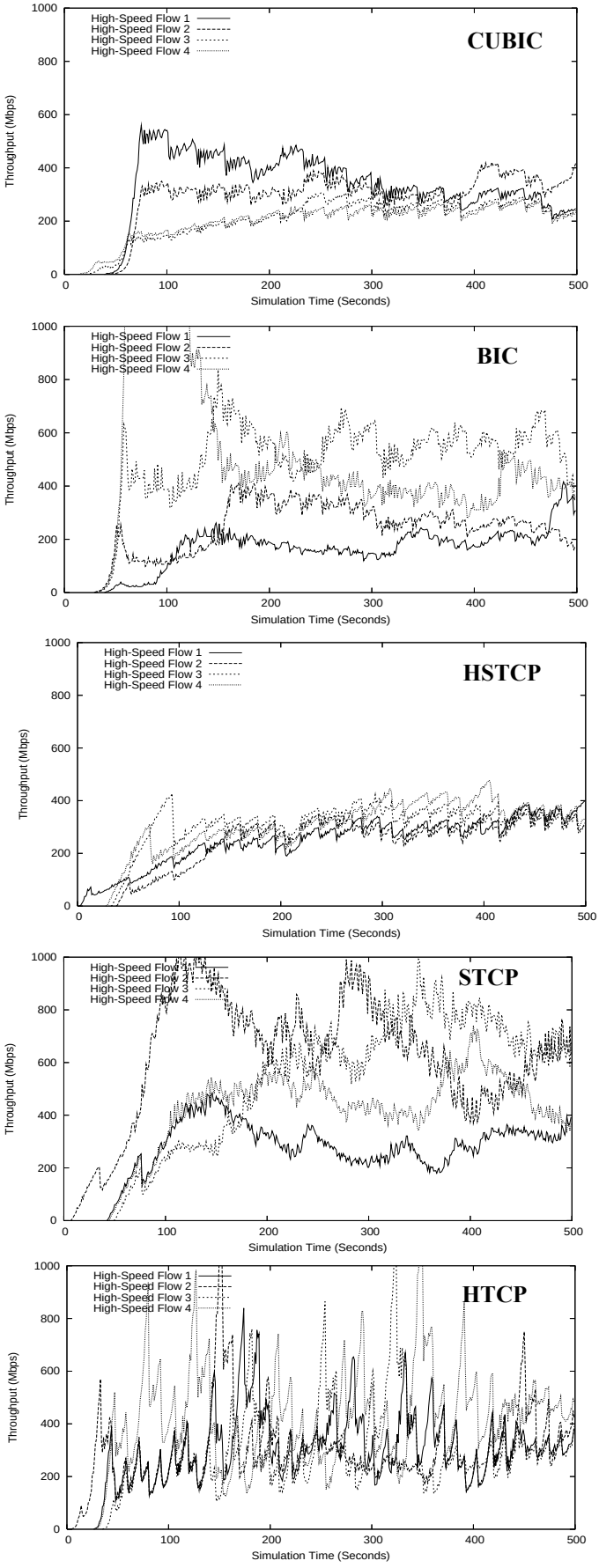

Fig. 9: Throughput of various protocols in stability test with 200% buffer

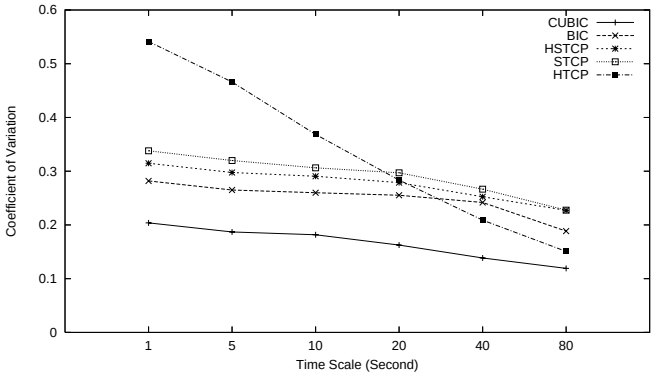

Fig. 10: CoV in stability test with 20% buffer

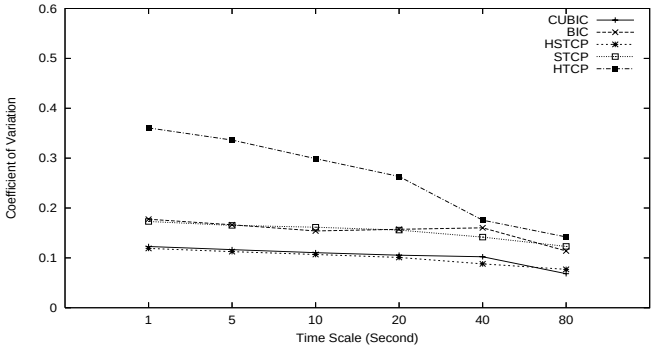

Fig. 11: CoV in stability test with 200% buffer

REFERENCES

[1] H. Bullot, R. Les Cottrell, and R. Hughes-Jones, "Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks," *Second International Workshop on Protocols for Fast Long-Distance Networks*, February 16-17, 2004, Argonne, Illinois USA

[2] C. Jin, D. X. Wei and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," In *Proceedings of IEEE INFOCOM 2004*, March 2004

[3] S. Floyd, "HighSpeed TCP for Large Congestion Windows," *RFC 3649*, December 2003

[4] T. Kelly, "Scalable TCP: Improving Performance in High-Speed Wide Area Networks," *ACM SIGCOMM Computer Communication Review*, Volume 33, Issue 2, pp. 83-91, April 2003

[5] R. Shorten, and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks," *Second International Workshop on Protocols for Fast Long-Distance Networks*, February 16-17, 2004, Argonne, Illinois USA

[6] T. Hatano, M. Fukuhara, H. Shigeno, and K. Okada, "TCP-friendly SQRT TCP for High Speed Networks," in *Proceedings of APSITT 2003*, pp455-460, Nov 2003.

[7] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," In *Proceedings of ACM Mobicom 2001*, pp 287-297, Rome, Italy, July 16-21 2001

[8] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," In *Proceedings of IEEE INFOCOM 2004*, March 2004

[9] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance," *Proceedings of IEEE INFOCOM 2004*, March 2004

[10] Yunhong Gu, Xinwei Hong, and Robert Grossman, "Experiences in Design and Implementation of a High Performance Transport Protocol," *SC 2004*, Nov 6 - 12, Pittsburgh, PA, USA

[11] Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer**,** **"**Equation-Based Congestion Control for Unicast Applications**,"** SIGCOMM 2000.