Sync-TCP: A New Approach to High Speed Congestion Control

Xiuchao Wu, Mun Choon Chan, A. L. Ananda and Chetan Ganjihal
School of Computing, National University of Singapore
Computing 1, #13 Computing Drive, Singapore 117417
Email:{wuxiucha, chanmc, ananda, chetan}@comp.nus.edu.sg

***Abstract*— As bandwidth in the Internet continues to grow, there will be more and more long fat network pipes with abundant residual bandwidth. At the same time, there is also a gradual and steady increase in the deployment of Internet endpoints equipped with different variants of high speed TCP.**

In this work, we first illustrate drawbacks associated with two widely deployed high speed TCP variants, namely: Cubic TCP and Compound TCP. We show that even with common and reasonable settings, problems can arise. Next, we present Synchronized TCP (Sync-TCP), a new delay-based high speed congestion control (HSCC) algorithm. The approach taken by Sync-TCP is novel in the following ways. First, Sync-TCP exploits synchronization. The key insight of Sync-TCP is that if competing flows could detect the same congestion signal through queue delay, these flows can coordinate their congestion control behaviors. Second, using only the basic mechanism, Sync-TCP will yield to legacy TCP when congestion is detected. Hence, Sync-TCP is designed to not hurt applications using legacy TCP or interactive applications.

We performed extensive simulation and some testbed evaluations to show that Sync-TCP achieves its design goals and it performs better than existing HSCC approaches including Fast TCP, Compound TCP and Cubic TCP, especially in the trade-off between throughput and friendliness.

I. Introduction

As bandwidth in the Internet continues to grow, there will be more and more long fat network pipes with abundant residual bandwidth. For example, expected capacity of the forthcoming Trans-Pacific Express is 5.12 Tbps, and FTTx (Fiber To The Home, Building, etc.) has been widely deployed in some countries. Recent bandwidth measurement statistics (*http://www.speedtest.net/*) show that the average download speeds in Europe and North America exceed 5Mbps. The average download speed for the top 6 countries already exceeds 10Mbps. In some countries, for example, Singapore, the goal is to provide up to 1Gbps broadband access by 2015.

However, it is well known that, TCP, the de-facto standard transport protocol of the Internet, cannot work well on long fat network pipe where the bandwidth-delay product (BDP) is large [1]. Due to the loss based AIMD (additive increase and multiplicative decrease) algorithm, legacy TCP versions (TCP Reno, TCP Newreno, TCP SACK, etc.) cannot send data fast enough. In order to address this problem, many high speed congestion control (HSCC) algorithms, such as Highspeed TCP [1], Cubic TCP [2], H-TCP [3], Fast TCP [4], TCP Illinois [5] and Compound TCP (CTCP) [6], have been proposed in recent years to better utilize the abundant residual bandwidth and provide higher throughput to end users.

While many of these proposals remain only as research prototypes, a small number of them have been gradually and steadily deployed on some widely available operating systems. In particular, consider the case of CTCP and Cubic TCP. CTCP is distributed with computers running on Windows Server 2008 and Windows Vista. Though disabled by default, CTCP can be easily enabled. As an indication of how widely available CTCP is, statistics show that 17% of all clients that visit the *www.w3schools.com* website used Windows Vista. For the case of Cubic TCP, it is the default configuration for Fedora 9 and Ubuntu 8.10 distributions. As of March 2009, there are more than 4 million unique IP addresses that have connected to download Fedora (9 and 10). The number of Ubuntu users would be large as well. Hence, it is clear that CTCP and Cubic TCP are sufficiently widely deployed and ready to fully exploit the growing availability in network bandwidth as more bandwidth becomes available. While this may be good news for the users of these HSCC algorithms, it is clear that the impact of using these HSCC algorithms on users of interactive traffic and legacy TCP should be urgently and thoroughly investigated.

In this work, we first identify issues that associate with the existing and widely distributed high speed TCP variants like CTCP and Cubic TCP. We show that even with common and reasonable settings, problems with efficiency and friendliness arise. Next, we present Synchronized TCP (Sync-TCP), a new delay-based HSCC algorithm.

The approach taken by Sync-TCP is novel in the following ways. First, Sync-TCP exploits synchronization. The key insight of Sync-TCP is that if competing flows could detect the same congestion signal through queue delay, these flows can coordinate their congestion control behaviors and drive the network to operate around the desired point. This is in contrast to the classic view that synchronization leads to bad performance since it is caused by traffic overload and competing flows simultaneously reduce their sending rate (blindly) by half [7]. Second, using only the basic mechanism, Sync-TCP will yield to legacy TCP when congestion is detected. Therefore, Sync-TCP is designed to not hurt applications using legacy TCP and interactive traffic since it backoffs earlier. We argue that a more conservative HSCC is a fair and appropriate behavior since a HSCC TCP variant has already been allowed

to utilize the excessive bandwidth. One can also think of Sync-TCP as operating as a best-effort service among the best effort services. When there is no or limited excessive bandwidth, Sync-TCP may also switch to legacy TCP to avoid starvation.

Sync-TCP is designed such that with high probability, competing flows can detect the same congestion signals through queue delay. Each Sync-TCP client uses an adaptive queue-delay-based congestion window decrease rule, and a RTT-independent congestion window increase rule designed to drive the network to operate around the knee [8] and to distribute the residual bandwidth fairly, independent of the number of flows in the bottleneck link and RTT heterogeneity among competing flows. When the network is operating around the knee, network throughput is high, queue delay is short, and packet drop rate is minimum. Hence, Sync-TCP is designed to be friendly not only to long-lived TCP flows in the metric of throughput, but also to the interactive applications in the metrics of delay and jitter.

We performed extensive evaluations using NS-2 simulations and some testbed evaluations. The results show that Sync-TCP achieves its design goals and performs better than existing HSCC approaches including Fast TCP, CTCP and Cubic TCP.

This paper is organized as follows. Section II introduces related work. The details of CTCP and Cubic TCP are then presented in section III. Some of their shortcomings are also highlighted. Section IV presents the details of Sync-TCP, and its deployment issues are discussed in section V. Simulation and testbed evaluation results are presented and analyzed in section VI and VII, respectively. Finally, conclusion is presented in section VIII.

II. Related Work

Scalable TCP [9], a loss based MIMD (Multiplicative Increase and Multiplicative Decrease) congestion control algorithm, is one of the earliest HSCC proposal for networks with large BDP. Since then, a lot of other HSCC algorithms have been proposed. Highspeed TCP [1] adopts some new window increase and decrease functions with the aim to converge quickly and avoid large burst of segment loss. Bic-TCP [10] adopts a concave $cwnd$ growth function for improving RTT fairness and reducing the number of packets dropped in one congestion event. Cubic TCP [2], an offspring of Bic-TCP, improves RTT fairness further by using a RTT-independent concave $cwnd$ growth function. In order to reduce packet loss rate suffered by cross traffic, TCP Illinois [5] slows down window increase rate when queue delay is increased. This technique is also adopted by Sync-TCP. H-TCP [3] adopts an adaptive AIMD algorithm for improving convergence and fairness. For improving RTT fairness, its additive increase factor is designed as a function of the clock time. Such a technique is also used in Sync-TCP.

All the above proposals decrease $cwnd$ only when segment loss is detected. Hence, the network tends to operate at the high utilization level and the cross traffic will unavoidably experience long queue delay. Based on this consideration, a number of delay-based HSCC algorithms are also proposed. However, these algorithms have a number of drawbacks as well. Fast TCP [4] does not work well when there are many competing flows since each flow tries to maintain a large number of packets in the queue of the bottleneck link. In the metrics of friendliness, CTCP [6] may be worse than Fast TCP as it needs to act as loss-based TCP for ensuring that its throughput is not less than the legacy TCP. Delay-based AIMD [11] tries to drive the network to operate around the knee independent of the number of competing flows. However, it does not address the issue of accurate delay measurement.

In delay-based HSCC algorithms, estimation of the RTPD (Round Trip Propagation Delay) is always a key issue. Such measurements are often complicated by in-flight packets in the network, rerouting, and the arrival and departure of flows. Sync-TCP is designed to address the issue of accurate queue delay measurement, flow scalability, RTT heterogeneity, and variations in cross traffic conditions.

In addition, there are also some non-high-speed delay-based congestion control algorithms, such as TCP Nice [12] and TCP-LP [13] that are proposed for low-priority background transfers. Sync-TCP is similar to these proposals in that Sync-TCP is designed to be "lower priority" than the legacy loss-based TCP. Similar to TCP Vegas [14], as these proposals increase $cwnd$ by at most one segment per round trip time, they are not suitable for bandwidth-greedy and elastic applications that run on long fat network pipes of the Internet.

Finally, some mechanisms, such as XCP [15] and VCP [16] that need supports from intermediate routers, have also been proposed for long fat network pipes. Sync-TCP is an end-to-end congestion control algorithm and does not assume and/or require network support.

III. Background and Motivations

Compound TCP (CTCP) and Cubic TCP are two HSCC algorithms that have been widely deployed. In this section, we briefly describe their behaviors to provide background information for evaluations and discussions in later sections. With a fairly common settings that mimics a SOHO (Small Office and Home Office) environment, the effects of the incremental deployment of CTCP and Cubic TCP are studied, and their limitations are pointed out.

A. Compound TCP [6]

CTCP runs two congestion control algorithms concurrently: the legacy TCP's AIMD algorithm and a delay-based HSCC algorithm. The sending rate of CTCP is determined by win which is the sum of $cwnd$ and $dwnd$. $cwnd$ follows the legacy TCP, and $dwnd$ is adjusted based on the delay-based HSCC algorithm. CTCP ensures that its throughput is never less than the legacy TCP by making $dwnd$ no less than 0.

$$\begin{aligned} \triangle &= (thr_{expected} - thr_{actual}) * brtt \\ &= (\frac{win}{brtt} - \frac{win}{srtt}) * brtt = \frac{win * (srtt - brtt)}{srtt} \\ &= thr_{actual} * (srtt - brtt) = thr_{actual} * qd \end{aligned} \quad (1)$$

In its delay-based HSCC algorithm, CTCP measures one RTT sample per millisecond (ms). At the end of each round trip time, $\triangle$ is calculated according to equation 1, which is widely used by delay-based congestion control algorithms. In this equation, $brtt$ is the minimum of all RTT samples, and $srtt$ is the arithmetic average of RTT samples measured in the current round trip time. Based on $\triangle$ and a global constant (γ), $dwnd$ is adjusted according to equation 2.

$$dwnd_{i+1} = \begin{cases} dwnd_i + (\alpha * win_i^k - 1)^+ & \triangle < \gamma, \\ (dwnd_i - \zeta * \triangle)^+ & \triangle \geq \gamma, \\ (win_i * (1-\beta) - cwnd/2)^+ & loss. \end{cases} \quad (2)$$

Here, $(\ \cdot\ )^+$ is defined as $max(\ \cdot\ ,\ 0)$. β is a constant and is now set to 1/2 so that CTCP will reduce sending rate by half when segment loss is detected. α and k are also constants whose values are selected so that when $\triangle$ is less than γ, CTCP increases win in a manner similar to Highspeed TCP [1]. When congestion is detected through queue delay, the reduction of $dwnd$ is related to $\triangle$, which is a little larger than γ. Hence, its delay-based HSCC algorithm is similar to a MIAD (Multiplicative Increase and Additive Decrease) algorithm and competing flows cannot converge quickly. Furthermore, considering that competing CTCP flows may act like legacy TCP at different times, their convergence behaviors can be very complex.

The number of packets that each CTCP flow tries to maintain in the queue of the bottleneck link fluctuates between ϕ and γ ($\phi < \gamma$). Here, ϕ is determined by ζ, and ϕ decreases with the increase of ζ. However, even when $\zeta > 1$, CTCP still cannot ensure that the queue of the bottleneck link can be emptied. According to equation 1, flows with higher throughput will detect congestion before flows with lower throughput. Although this method can speed up convergence, competing flows will not detect congestion simultaneously. Hence, competing flows cannot reduce $dwnd$ simultaneously, and consequently the queue of the bottleneck link cannot be emptied and new flows cannot estimate their RTPD correctly.

In CTCP, the threshold γ should not be too small. From equation 1, when thr_{actual} is high and γ is small, the noise in RTT samples can be mistaken as congestion signals, resulting in unnecessary reduction of sending rate. On the other hand, with a large γ, a small number of CTCP flows can cause buffer-overflow before congestion could be detected through queue delay. As the rate of $dwnd$ increase is related to win and can be very large on long fat network pipes, many packets can be dropped in a congestion event and cross traffic will be drastically affected.

B. Cubic TCP [2]

Cubic TCP is an offspring of the novel Bic-TCP [10]. With the aim to improve RTT fairness, similar to H-TCP [3], the window growth function of Cubic TCP is also designed to be independent of RTT. Equation 3 shows the exact rule used by Cubic TCP for updating its window size. Here, C is a scaling factor, t is the elapsed time since the last window reduction, β is a constant multiplicative decrease factor applied when segment loss is detected, and W_{max} is the window size just before the last window reduction.

$$W_{cubic} = C(t - \sqrt[3]{W_{max} * \beta / C})^3 + W_{max} \quad (3)$$

Hence, the closer W_{cubic} is to W_{max}, the smaller its increase step is. Ideally, the concavity of its window growth function can effectively reduce the number of segments dropped in a congestion event.

However, experimental evaluation had identified several limitations of Cubic TCP, such as slow convergence and prolonged unfairness [17]. Furthermore, Cubic TCP is fundamentally a loss-based HSCC algorithm, which probes network resources aggressively to maintain short congestion epoch (the time between two consecutive congestion events) on long fat network pipes, and reduces sending rate only when segment loss is detected. Hence, the network tends to operate at the high utilization level and the cross traffic will unavoidably experience long queue delay.

C. SOHO Environment Study

A SOHO setting is shown in figure 1. The access link of the SOHO network is 20Mbps. DropTail is used by the access router and queue size is set to 0.5BDP. Round trip time between the users and hosts of the Internet is simulated as 220ms, and packet loss rate of the current Internet is simulated as 10^{-4}. The simulation time is 1040 seconds.

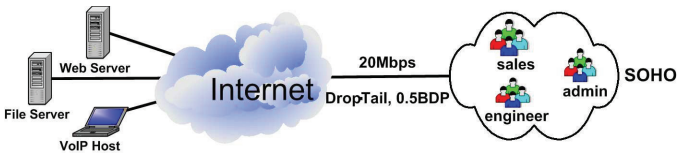


Fig. 1. SOHO Network

As for network traffic, one UDP flow is used to simulate a salesman talking to a client through VoIP (Voice over IP). Using the connection-based web traffic model [18], 10 web transactions are generated per second to simulate employees browsing the Internet. These traffics exist throughout the simulation and they are used to collect user experience of VoIP and web surfing.

Five long-lived flows are used to simulate downloading activities. In the beginning, all long-lived flows are driven by legacy TCP with small buffer (64KB). This is the situation before HSCC algorithms are deployed. We then let 1, 2, 3 or 4 flows adopt a HSCC algorithm with the aim to study the effect of this HSCC algorithm's incremental deployment. All of the five long-lived flows start at the 20th second and they end at the 1020th second.

The simulation results are plotted in figure 2. As link utilization ratio of the access link is close to 1 in all cases, we only plot the response time of HTTP transactions whose response size is 16KB, the throughput of a FTP flow driven by legacy TCP with a 64KB socket buffer, and one way delay

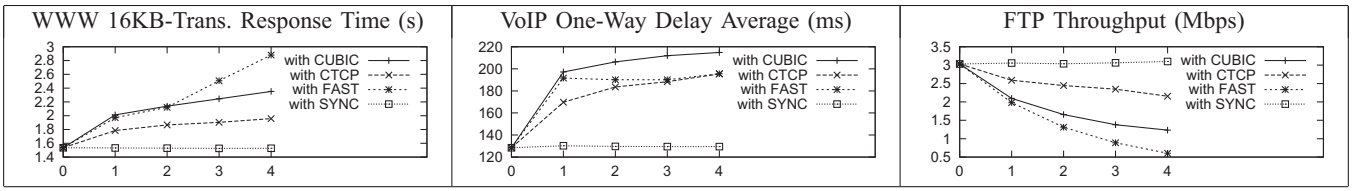


Fig. 2. User Experience of Cross Traffic Applications With the Incremental Deployment of HSCC Algorithms in a SOHO Environment

experienced by VoIP packets. Based on these results, we can see that:

- Cubic TCP is too aggressive. Even when it is adopted by only 1 flow, it significantly reduces throughput of legacy FTP flow (30%) and substantially increases the delay suffered by VoIP packets (56%). When more flows adopt Cubic TCP, the situation becomes even worse. When 4 flows adopt Cubic TCP, HTTP response time for a 16KB download increases by 33%.
- Although CTCP performs like TCP in this SOHO network with a 20Mbps bandwidth, when there are only 1 CTCP flow, user experience of VoIP, web surfing, and Legacy FTP still degrade by 32%, 20% and 13.3% respectively.
- Fast TCP [4], another influential HSCC algorithm, performs even much worse than CTCP and Cubic TCP. Although Fast TCP is a delay-based HSCC algorithm, each Fast TCP flow tries to maintain a large number of packets in the queue of the bottleneck link. Hence, competing Fast TCP flows cannot detect congestion through queue delay before buffer overflow occurs, many segments are dropped, and cross traffics are hurt. When 4 flows adopt Fast TCP, user experience of VoIP, web surfing, and Legacy FTP degrades by 60%, 93% and 80% respectively.
- Our proposal, Sync-TCP, performs the best with minimum impact on VoIP, web and FTP applications. The results indicate that the incremental deployment of Sync-TCP will not hurt other applications. As we will shown later in sections VI and VII, Sync-TCP can keep this friendliness to cross traffic and utilize the excess bandwidth efficiently in most of the scenarios evaluated.

IV. Design of Sync-TCP

Synchronized TCP (Sync-TCP) is fundamentally a delay-based HSCC scheme that tries to drive long fat network pipes of the Internet to operate around the knee of the delay-throughput curve [8]. Sync-TCP is firstly designed such that competing flows can detect the same congestion signals through queue delay. While the term *synchronize* is used to describe how competing flows should detect a congestion signal through queue delay, all Sync-TCP needs is a consistent view. Hence, Sync-TCP only needs the *congested state* to last for a sufficiently long period so that competing flows have high probability of detecting the same congestion state.

According to simple analysis [19] and a stochastic matrix model [20], AIMD algorithms can ensure that competing flows could converge quickly and share network resources fairly in synchronized communication networks. Considering that Sync-TCP has already been designed to make queue delay to be a highly synchronized congestion signal, it is reasonable for Sync-TCP to adopt an AIMD algorithm. The details of Sync-TCP are presented as follows.

A. Queue Delay Measurement and Congestion Detection

1) RTT Sampling: To save system resources in high speed networks, instead of sampling on each ACK, Sync-TCP only measures RTT once per T_{sample}. Pacing is also adopted so that a sender can sample queue delay evenly and avoid the large RTT noise caused by Delayed ACK [21]. The granularity of pacing timer (T_{pace}) should be smaller than T_{sample}, and both of them should be much smaller than round trip time of a long fat network pipe.

With an event-based pacing scheme [22], the overhead of TCP pacing can be reduced significantly. According to our implementation in FreeBSD and third party's implementation in Linux [4], CPU processing overhead only increases slightly when T_{pace} is several millisecond.

2) $srtt$, $brtt$, and $brtt_{epoch}$ Updating: The exponentially smoothed average of RTT samples, $srtt$, is updated based on equation 4 for filtering out the noise.

$$srtt_{i+1} = (1 - \frac{T_{sample}}{T_{win}}) * srtt_i + \frac{T_{sample}}{T_{win}} * rtt_i \quad (4)$$

Here, T_{win} is a global constant so that competing flows could have a consistent view of network state independent of their RTPD values. For each RTT sample, $brtt$ and $brtt_{epoch}$ are also updated based on the following equations.

$$brtt = min(brtt, rtt_i), brtt_{epoch} = min(brtt_{epoch}, rtt_i)$$

In Sync-TCP, $brtt$ is the estimation of RTPD and is set to the minimum of all RTT samples. As for $brtt_{epoch}$, it is the minimum of RTT samples observed in the current congestion epoch. In Sync-TCP, congestion epoch is defined as the period between two consecutive $cwnd$ reductions. When $cwnd$ is reduced, $brtt_{epoch}$ will be used to judge whether previous $cwnd$ reduction is large enough to empty the queue of the bottleneck link.

3) Congestion Detection: Sync-TCP calculates qd (the difference between $srtt$ and $brtt$) and compares it with Th_{qd}, a global constant. If $qd > Th_{qd}$, Sync-TCP detects a congestion signal through queue delay. Section IV-E will discuss how the value of Th_{qd} should be selected.

Note that since Sync-TCP determines network state per RTT sample and pacing is adopted, the interval between the nearest points that competing flows determine network state should

not be much larger than T_{sample}. Hence, competing flows has the potential of detecting congestion at almost the same time.

B. Synchronization of Congestion Detection

A key feature of Sync-TCP that enables synchronization of congestion detection is the delayed decrease after congestion detection and the delayed increase after $cwnd$ reduction.

After a congestion signal is detected through queue delay, instead of reducing $cwnd$ immediately, $cwnd$ is freezed for a short period (T_{wait}) before it is reduced. In this *waiting* period, RTT is sampled, $brtt$ & $brtt_{epoch}$ & $srtt$ are updated, but queue delay based congestion detection is not carried out.

This delayed $cwnd$ reduction is introduced for two reasons. First, queue delay is a *delayed* network feedback. With this delayed $cwnd$ reduction, end hosts can reduce $cwnd$ based on the real queue delay caused by the *freezed* $cwnd$. Second, by keeping network load constant and high for a period, it is unlikely that competing flows will miss this congestion signal.

Similarly, after $cwnd$ is reduced, instead of increasing $cwnd$ immediately, Sync-TCP will also freeze the just reduced $cwnd$ for T_{wait} so that the bottleneck link could have some time to empty packets previously buffered in its queue. In this *emptying* period, the behaviors of Sync-TCP are identical to the behaviors in the above *waiting* period. This period is introduced so that competing flows can have a more accurate measure of $brtt$. Due to the following queue-delay-based $cwnd$ decrease rule, these *emptying* periods will not lead to much lower link utilization ratio as there are still sufficient packets in the network.

C. Adaptive Queue-delay-based cwnd Decrease Rule

When $cwnd$ is to be reduced due to queue delay measurement, Sync-TCP calculates β, the multiplicative decrease factor, based on equation 5. Here, $srtt_{reduce}$ is the value of $srtt$ when $cwnd$ is reduced and qd_{reduce} equals to the difference between $srtt_{reduce}$ and $brtt$. Lower and upper bounds are also used for safety.

$$\beta = 1 - \frac{\lambda * qd_{reduce}}{srtt_{reduce}}, 0.125 \leq \beta \leq 0.95 \qquad (5)$$

In order to adapt to variations in cross traffic conditions, β needs to be adaptive so that the queue of the bottleneck link can be emptied periodically. More discussions about this issue can be found in [23]. To achieve this purpose, λ is first initialized to a small constant (1.25), and its value is adjusted as follows.

When $cwnd$ is reduced, Sync-TCP first calculates the difference between $brtt_{epoch}$ and $brtt$. If the difference is larger than $Th_{emptied}$, it indicates that β calculated with the current λ cannot empty the queue of the bottleneck link. Hence, λ is increased by one. This large increase step is used to ensure that the queue can be emptied in a short time. If the difference is less than $Th_{emptied}$, it indicates that the current λ is large enough and it might need to be decreased. Since we cannot deduce how much the current λ is larger than its optimal value, λ is reset to 1.25. The value of $Th_{emptied}$ is a function of how much error is expected in the delay measurement due to the noise in RTT samples. In this work, $Th_{emptied}$ is set to 2ms.

After $cwnd$ is reduced, $brtt_{epoch}$ is set to a huge value for tracking the minimal RTT sample that will be observed in the following congestion epoch.

For safety, Sync-TCP will also reduce $cwnd$ when segment loss is detected. In order to avoid under-utilizing lossy and fast links, unlike Fast TCP and CTCP that reduce sending rate by half, β is also calculated based on equation 5. However, as segment loss may be a signal of severe network congestion, the maximum of β is reduced to 0.875 and $cwnd$ is reduced immediately without delay.

D. RTT-Independent cwnd Increase Rule

In Sync-TCP, the additive $cwnd$ increase rule is designed with the following considerations. First, to work well on long fat network pipes, α (the additive increase factor) should increase with the elapse of time. The reason is that if congestion is not detected after an extended period of time, there are either very few competing flows or there are substantial excess bandwidth, and $cwnd$ should be increased more quickly. Second, to distribute bandwidth fairly among competing flows independent of their RTPD values, α should be calculated by a RTT-independent function. Furthermore, instead of increasing $cwnd$ by α segments per round trip time, it should be increased by α per *fixed-length* period so that all competing flows will increase $cwnd$ by the same amount of segments in the same period. A general form of how α should be increased can be written as follow:

$$\alpha = a_0 + a_1 * t + a_i * t^i, (i \geq 2).$$

Here, t is the clock-time elapsed since $cwnd$ started increasing, in the unit of second. α is first increased slowly (through the constant and linear term t) for safety. With the elapse of time, α increases very faster depending on the values of a_i and i. A similar form has been adopted by H-TCP [3], where a_0, a_1, i and a_i are set to 1, 10, 2, and $\frac{1}{4}$ respectively.

In our work, Sync-TCP increases $cwnd$ by α segment per T_{win}, and α is calculated based on equation 6.

$$\alpha = \max((1 + t + \frac{t^4}{32}) * \frac{Th_{qd} - qd}{Th_{qd}}, 1) \qquad (6)$$

Hence, a_0, a_1, i and a_i are set to 1, 1, 4, and $\frac{1}{32}$ respectively in Sync-TCP. In order to probe the network bandwidth quickly, the higher order term t^4 is used and by setting a_4 to $\frac{1}{32}$, the higher order term dominates over the linear term t when t increases beyond 2.4s. Note that the exact coefficients and power of these terms are not crucial for the correctness but do affect the aggressiveness of $cwnd$ increase and the speed of convergence. The current values are selected based on simulation results under various scenarios.

With the consideration that queue delay is a *delayed* network feedback, $\frac{Th_{qd}-qd}{Th_{qd}}$ is used here to slow down the increase of α as qd approaches Th_{qd} such that the maximal queue delay will not be much larger than Th_{qd}.

E. Parameter Selection Guidelines

There are only a small number of parameters that play key roles in the Sync-TCP. The other parameters can be deduced based on these global parameters and/or host-specific configuration.

1) Th_{qd} *:* Congestion detection is based on Th_{qd} and its value reflects the amount of buffering desired at the bottleneck link (independent of the number of competing flows). There are two considerations for choosing Th_{qd}. First, if qd_{max} is the largest queue delay that cross traffic applications can tolerate, Th_{qd} should be less than qd_{max} since queue delay is a delayed network feedback. Second, Th_{qd} should also be large enough to avoid regarding the noise in RTT samples as the signal of network congestion.

Considering that average one-way jitter experienced by the high quality VoIP is targeted at less than 30ms and VoIP packets may pass through multiple congested links, qd_{max} is set to 20ms and Th_{qd} is currently set to 12ms.

2) T_{win}*:* Another global variable of Sync-TCP is T_{win}. As shown in equation 4, when the sender updates $srtt$, T_{win} is used so that competing flows could have a consistent view of network state independent of their RTPD values. T_{win} must be large enough so that enough samples are considered, the noise in RTT samples can be filtered out, and $srtt$ can correctly reflect queue dynamics in the last T_{win} period. On the other hand, if T_{win} is too large, response to changes in queue dynamics will be affected.

The value of T_{win} is determined in the following way. First, as T_{sample} determines the amount of processing and memory overhead required, we assume that a value of 10ms can be supported by most endpoints without imposing excessive load. Next, a sample size of at least 10 within a window of T_{win} is assumed to be needed. Hence, the value of T_{win} is set to 100ms, so that there are at least 10 samples, with at least one sample every 10ms.

3) T_{wait}*:* This global variable is used for synchronizing congestion signal and reducing $cwnd$ based on the real queue delay caused by the frozen $cwnd$. In order for Sync-TCP to function correctly, T_{wait} must be long enough so that all Sync-TCP endpoints detect this synchronization signal.

For this to be the case with high probability, T_{wait} should be larger than $RTT_{max} + \kappa * T_{win}$. Here, RTT_{max} is the longest RTT experienced by all Sync-TCP flows and κ is a small number (no less than 1). This is needed so that all endpoints will detect the congestion signal with sufficient confidence. With the consideration of the possible maximal length of a network path in a territorial network and the light speed in wire, T_{wait} is currently set to 500ms.

Among the three global variables, Th_{qd} must be followed by all senders. As for T_{win} and T_{wait}, Sync-TCP still can work if the senders adopt slightly different values, but the fairness among their flows will be affected.

F. State Transition in Sync-TCP

Based on the above description, Sync-TCP can be summarized by the state transition diagram shown in figure 3. In all states of Sync-TCP (*Probing*, *Waiting*, and *Emptying*), RTT is sampled per T_{sample}, and for each RTT sample, $srtt$ & $brtt$ & $brtt_{epoch}$ are updated. $cwnd$ is increased and queue-delay-based congestion detection is carried out only in the *Probing*-state. When Sync-TCP leaves the *Waiting*-state or segment loss is detected in any state, $cwnd$ is reduced and $brtt_{epoch}$ is set to a huge value in order to track the smallest RTT sample to be observed in the following congestion epoch.

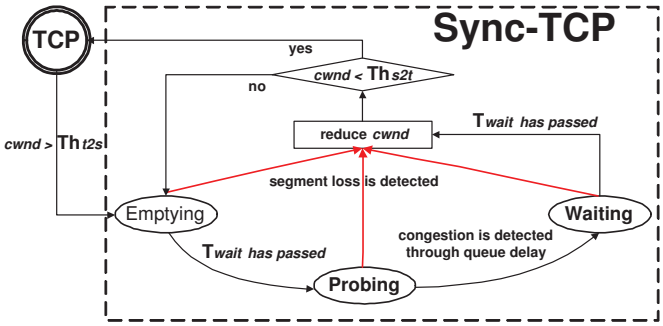


Fig. 3. State Transition Diagram of Sync-TCP

V. Discussion

As shown in figure 3, an addition feature of Sync-TCP is that a flow may switch between Sync-TCP and legacy TCP. This design is made based on the following considerations. As for parameter selection guidelines for the switching thresholds (Th_{t2s} and Th_{t2s}), please refer to the report [23].

When a flow begins to transmit, it does not know the size of its network pipe. Hence, the legacy TCP mode should be used first and Sync-TCP is activated only when the flow believes that it is on a long fat network pipe and there is sufficient excessive capacity. As a delay-based congestion control, the throughput of a Sync-TCP flow should be high so that there are enough RTT samples to measure queue delay correctly. Hence, when throughput is too low, Sync-TCP should also switch back to the legacy TCP. In addition, switching back to TCP is also helpful to solve the following deployment issues.

A. Coexisting with Loss-Based Flows

Taking into account how Sync-TCP detects congestion and handles segment loss, it should be obvious that when the network utilization is high because of the increased legacy TCP applications and interactive applications, Sync-TCP flows may be starved and per-flow throughput becomes low. In this scenario, letting flows of bandwidth-greedy and elastic applications act as legacy TCP is fairer and provides more incentives to adopt Sync-TCP.

We have investigated the issues of coexisting with TCP flows that are well tuned for high speed data transmission. When the load of cross traffic fluctuates with time, Sync-TCP can acquire comparable bandwidth even when packet error rate is very low. The reason is that when the load of cross traffic is reduced, Sync-TCP can acquire the suddenly increased bandwidth more quickly.

As for the coexistence with loss-based HSCC algorithms, Sync-TCP will acquire much less bandwidth since it responds to the earlier congestion signal, queue delay. Standardization and/or queue management mechanisms, such as ZL-RED [24] that catches and punishes loss-based flows, should be adopted for solving these issues.

B. Multiple Congested Links

Since Sync-TCP detects congestion by comparing queue delay with a constant, when a Sync-TCP flow passes through multiple congested links (MCL), it can be starved by other flows which only pass through one of these congested links.

We argue that it is reasonable since the flow that passes through multiple congested links consumes more network resources for transmitting the same amount of data. Furthermore, when a flow passes through multiple congested links and the throughput becomes very low, it will switch back to TCP. As a result, the flow will not be totally starved.

C. Handling Rerouting

In Sync-TCP, $brtt$ is set to the minimum of all RTT samples. However, if RTPD is increased due to the change of network path, RTPD cannot be correctly estimated, and when $cwnd$ is reduced, λ will always be increased and β will be very small.

In the case that there are other competing flows, this flow will be starved and it will switch back to TCP. When switching to TCP, Sync-TCP will set $brtt$ to a huge value so that this flow can correctly learn the increased RTPD. If there is no other competing flows or all flows experience rerouting simultaneously, this flow may not switch back to TCP since β has a lower bound. But its sending rate will keep fluctuating and the bottleneck link can be slightly under-utilized. Hence, if λ is huge (> 20) and $brtt_{epoch}$ is still larger than $brtt$, Sync-TCP will set $brtt$ to $brtt_{epoch}$ and λ is reset to 1.25.

In the following two sections, we evaluate Sync-TCP through both simulation and testbed evaluation. In the simulation, we can experiment with more high speed TCP variants and more variations in traffic load, in particular, a much larger number of HSCC and cross traffic flows. On the other hand, we have a more realistic environment in the testbed evaluation, but the availability of high speed TCP variants and the scalability of the traffic load are much lower.

VI. Simulation Results

In the simulation, Sync-TCP has been implemented in the framework of NS-2 TCP-Linux [25] so that we can compare it with HSCC algorithms that have been implemented in Linux, such as Cubic TCP. The leaky-bucket-based TCP pacing algorithm proposed in [22] is implemented for Sync-TCP and Fast TCP. CTCP and Fast TCP are also implemented according to their initial proposals, and their parameters are set to default values. In the simulation, Sync-TCP is evaluated and compared with Cubic TCP, CTCP, and Fast TCP. The legacy TCP implemented in Linux is also compared under the assumption that socket buffer is large enough and window scale option is enabled for supporting high speed data transmission.

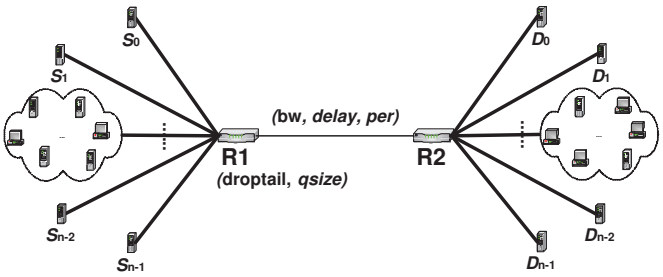


Fig. 4. Dumbbell Network Topology

The simulation topology used is the dumbbell network configuration shown in figure 4. The link between $R1$ and $R2$ is the simulated bottleneck whose bandwidth is 1Gbps and DropTail is used by the routers. Side links that connect S_i to $R1$ or connect D_i to $R2$ are all highly reliable and fast enough so that $R1 \leftrightarrow R2$ is the only bottleneck. In all experiments, the flows from S_i to D_i are the flows to be investigated.

Nodes in the two "clouds" are responsible for generating background traffic. The propagation delays to the router R1 (or R2) follow an uniform distribution whose range is [5ms,25ms]. According to the connection-based web traffic model [18], the two clouds generate 3200 (800) HTTP sessions per second in the forward (reverse) path. For collecting user experience of the cross traffic applications, there are also several VoIP flows and several legacy FTP flows which are driven by the legacy TCP and are configured with small socket buffer (64KB). These background traffic consumes about 300Mbps (100Mbps) in the forward (reverse) path. Unless stated otherwise, this background traffic is generated in all simulations presented in this section.

A. Synchronization of Congestion Signals

In order to understand how well competing Sync-TCP flows observe a consistent view of network state and detect congestion simultaneously through queue delay, we record the following information in the simulation log files. For each queue-delay-based congestion signal, we record the time that this signal is first detected by a flow. We then count the number of flows that also detect congestion through queue delay within the following T_{wait} period. If this number equals to the number of current active flows, this congestion signal is considered to be detected by all competing Sync-TCP flows.

We recorded observations over a total simulation time of about 9 hours from various experiments. In total, 5853 congestion signals are detected by Sync-TCP flows through queue delay and 5651 (96.6%) of these signals are detected by all competing Sync-TCP flows. In some of these experiments, the number of competing Sync-TCP flows are large (e.g. 40, 64, or 256) and their RTPD values differ significantly. 202 (3.4%) of the congestion signals are not detected by all competing Sync-TCP flows because some flows have just experienced segment loss and are in the *Emptying* state, in which queue-delay-based congestion detection is not carried out. However, note that in such cases, the number of Sync-TCP flows that

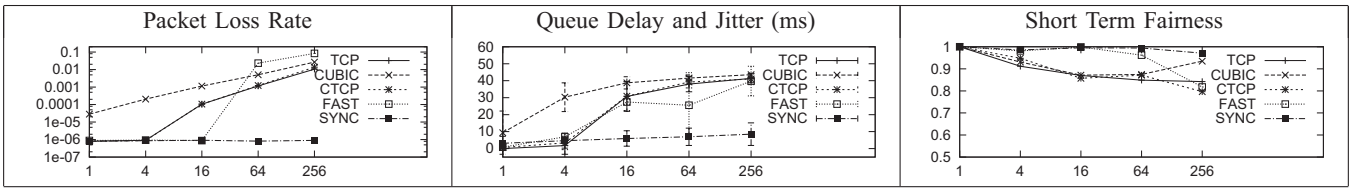


Fig. 5. Scalability with Flow Number

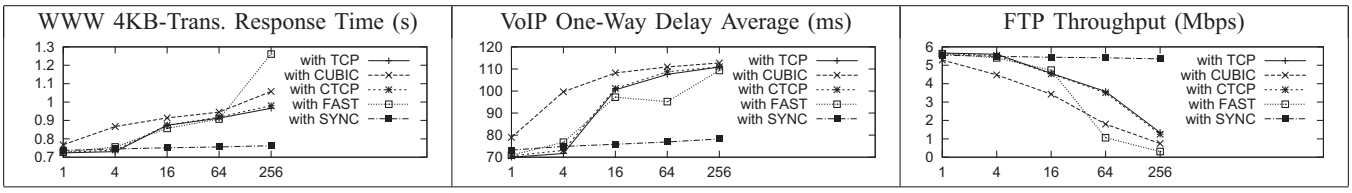


Fig. 6. Scalability with Flow Number: User Experience of Cross Traffic Applications

miss the congestion signal is small since packet loss rate is low when the delay-based Sync-TCP is adopted.

B. Flow Number Scalability

In this experiment, the goal is to evaluate how well these proposals perform with increasing number of flows. Dumbbell topology shown in figure 4 is used, and the arrival and departure of competing flows follow the block scenario shown in figure 7. The bottleneck link is configured as $delay$(propagation delay)=50ms, per(packet error rate)=10^{-6}, and $qsize$(queue size)=0.5BDP. Propagation delay of the side links are all set to 5ms. The number of HSCC flows, N, is then set to 1, 4, 16, 64, and 256. In order to minimize the startup effect, performance measurements are taken only after 100s.

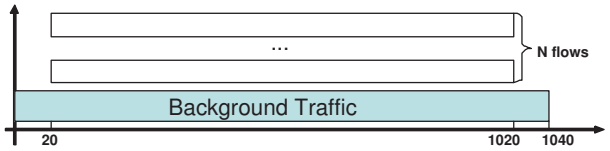


Fig. 7. Block Scenario: the Arrival and Departure Sequence of Flows

Figure 5 shows packet loss rate and queue delay at the bottleneck link. As the link utilization ratios are close to 1 in all cases except for TCP, the results are not shown. The results show that Sync-TCP can drive the network to operate around the knee, where loss and delay is small, independent of the number of competing flows. As a result, cross traffic applications will not be hurt. Figure 6 shows the user experience of the VoIP, web surfing, and legacy FTP traffic. With 64 flows running other HSCC algorithms, compared to Sync-TCP, the response time for web transactions and one way delay for VoIP traffic are at least 25% longer, and the throughput for legacy FTP traffic is at least 30% lower. The differences increase when the number of flows increases to 256. This result clearly demonstrates Sync-TCP's unique property of using only excessive bandwidth and having minimal impact on cross traffic. Figure 5 also shows short-term fairness index, which is the average of Jain's fairness indexes calculated in one-second intervals. The result shows that only Sync-TCP can provide fairness over small intervals over the entire range.

We also evaluated the case of 1024 flows. In this case, per-flow throughput becomes very low and Sync-TCP is not enabled. If there is no switching and Sync-TCP is always enabled, Sync-TCP can keep its friendliness to cross traffic even when $N = 1024$.

Scalability of Sync-TCP with respect to propagation delay, queue size, and packet error rate have been evaluated too. Sync-TCP performs very well in all cases except when queue size is very small, such that the maximum queue delay is much less than Th_{qd}, which is now set to 12ms. In this case, Sync-TCP cannot detect congestion through queue delay, $\frac{Th_{qd}-qd}{Th_{qd}}$ cannot effectively reduce α when buffer overflow approaches, and many segments are dropped in each congestion event. It may be worthwhile to let a flow switch between Sync-TCP and Cubic TCP based on whether it can observe queue delay that is larger than Th_{qd}.

C. RTT Fairness

Dumbbell topology and block scenario are also used here. N is set to 2. As for the bottleneck link, $delay$=20ms, per=10^{-6}, and $qsize$=0.5BDP. Propagation delay of side links are set to different values so that RTPD of flow 0 is always 60ms and RTPD of flow 1 is 60, 120, 240, 360, or 480ms in different experiments.

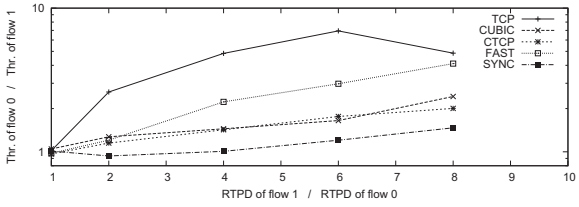


Fig. 8. RTT Fairness

Figure 8 shows the throughput ratio ($\frac{\text{Thr}_0}{\text{Thr}_1}$) against the RTPD ratio ($\frac{\text{RTPD}_1}{\text{RTPD}_0}$). It indicates that Sync-TCP performs the best in the metric of RTT fairness. Sync-TCP is able to distribute

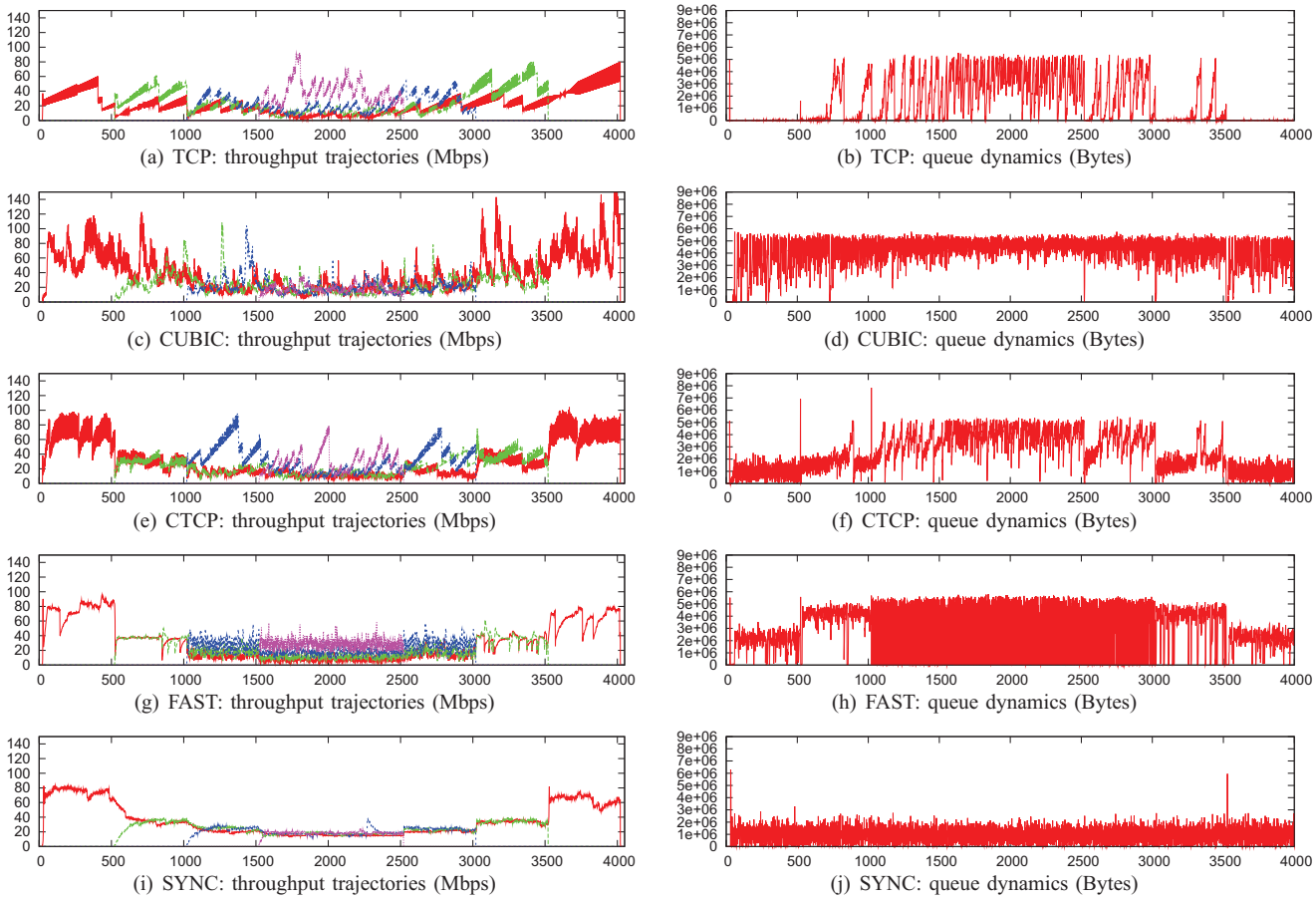


Fig. 9. Dynamic Scenario: throughput trajectories of flow 0, 10, 20, 30 (left) and queue dynamics of the bottleneck link (right)

bandwidth quite fairly between the two competing flows even when their RTPDs differ significantly. As for CTCP, it uses slow start of TCP, and when the first congestive segment loss occurs, the sending rate of flow 1 is very small (due to its long round trip time). After that, the two CTCP flows converge very slowly due to its MIAD-like delay-based HSCC algorithm of CTCP, and flow 1 keeps receiving much less throughput. Results are similarly bad for Cubic TCP, and Fast TCP performs even worse.

D. Dynamic Scenario

In the following experiments, dumbbell topology is used, and the bottleneck link is configured as $delay$=50ms, per=10^{-6}, and $qsize$=0.5BDP. Flow arrival and departure sequences shown in figure 10 are used for evaluating these proposals. In this scenario, there are 40 flows that are active over different periods and have different RTT values.

Figure 9 shows throughput trajectories of several HSCC flows that belong to different groups. The results indicate that, only Sync-TCP flows can quickly converge to the new bandwidth allocation equilibrium when flows arrive or leave. In equilibrium, Sync-TCP flows receive the same throughput irrespectively of the values of their RTPD.

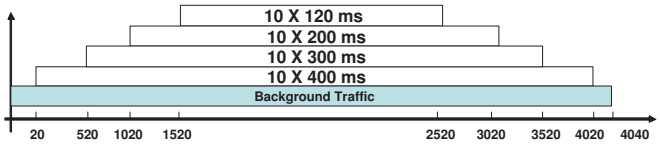


Fig. 10. Flows Arrival and Departure Sequence of Dynamic Scenario

Figure 9 also shows queue dynamics of the bottleneck link. The plots for link utilization ratio are close to 1 in all cases except for TCP and are not shown here. The results show that Sync-TCP is the only one that can maintain low queuing delay and high link utilization independent of the number of competing flows, the values of their RTPD, and the arrival and departure of flows.

Apart from the above simulations, many experiments, which use different parameter values (bw, per, $delay$, $qsize$, N), different flow arrival and departure sequences, and different cross traffic loads, have also been carried out. They are not presented here due to the similarity of these results. In addition, we also evaluated the impact of rerouting. When RTPD is increased due to rerouting, only Sync-TCP flows can converge to the fair and efficient point again.

VII. Testbed Evaluation Results

In order to evaluate Sync-TCP in more realistic environments, Sync-TCP has been implemented in FreeBSD 7.1, and a high speed network testbed has also been set up. In this section, we first introduce configuration of the testbed. Experiment results are then presented and analyzed.

A. High Speed Network Testbed

Figure 11 illustrates the topology and configuration of our high speed network testbed. A HP Procurve 2900 switch, which is configured with two VLANs (Virtual LAN), is used to connect all computers. Three high end Dell PowerEdge T300 servers (SENDER, EMULATOR, and SINK), which are equipped with Myricom 10Gbps ethernet cards, are connected to the HP Procurve's four 10Gbps ports.

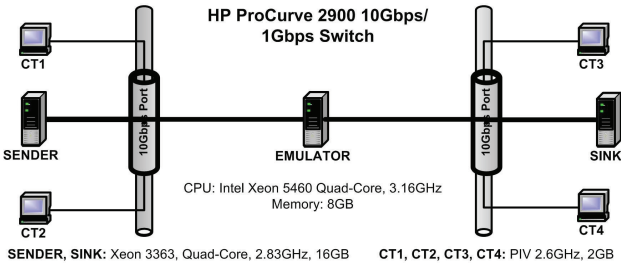


Fig. 11. High Speed Network Testbed

SENDER is installed with FreeBSD 7.1, Fedora 9, and Windows Vista so that we can compare CTCP on Windows Vista, Cubic TCP on Linux, and Sync-TCP on FreeBSD. Iperf [26] is used by SENDER and SINK to generate long-lived flows and collect statistics. In order to emulate the networks to be studied, EMULATOR is installed with FreeBSD and Dummynet [27], and the FreeBSD kernel is rebuilt with a higher scheduling frequency for emulating a high speed network accurately. DropTail is emulated by EMULATOR in all experiments. As for SINK, FreeBSD and Dummynet are installed. The Dummynet on SINK is used to emulate flows with different RTPDs.

Four lower end computers are connected to 1Gbps ports. They use D-ITG [28] to collect user experience of VoIP. Iperf is also used to emulate legacy FTP flows with a small socket buffer (64KB). These computers are also used to generate large amount of bursty traffic for emulating web-like traffic.

When setting up the following experiments, we make sure that there is enough memory for all flows and that the CPU is not the bottleneck. Hence, the number of flows per machine does not exceed 30, and the emulated bandwidth is not higher than 1Gbps even though the link rate is 10Gbps.

B. Synchronization of Congestion Signal

To verify whether Sync-TCP flows can detect congestion correctly in a testbed setting, three Sync-TCP flows with different RTPDs (90ms, 100ms, and 110ms, respectively) are established between SENDER and SINK. EMULATOR emulates a bottleneck link whose bw=500Mbps, $delay$=25ms, per=10^{-6}, and $qsize$=0.5BDP. Web-like background traffic consumes about 40Mbps, and there are also a VoIP flow and a legacy FTP flow.

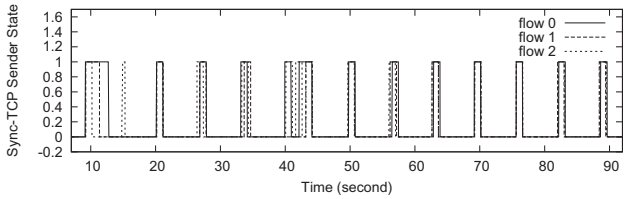
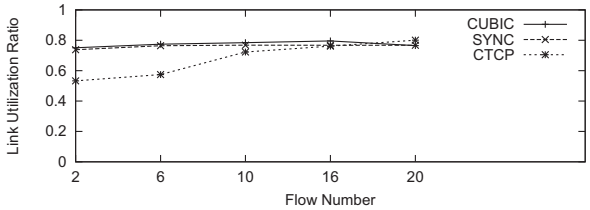


Fig. 12. Synchronization of Congestion Detection through Queue Delay

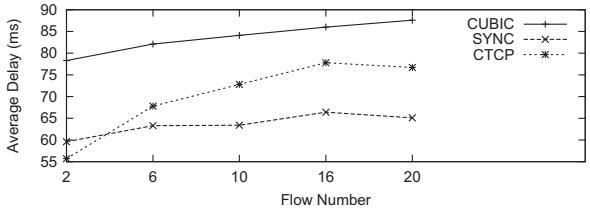
The length of this experiment is about 90 seconds. Figure 12 plots the changes of the three competing Sync-TCP flows' state with the time. "0" is used to represent *Probing* state, "1" is used to represent *Waiting* and *Emptying* states, and one pulse represents that a flow detects one congestion signal. Figure 12 indicates that except at the 15th and 42th seconds, all flows detect the same congestion signals. When background traffic is bursty and packet corruption is emulated, a small amount of congestion misses is expected. According to log data at EMULATOR, the queue of the bottleneck link is indeed emptied periodically, allowing accurate delay measurements.

C. Flow Number Scalability

In this set of experiments, the bottleneck link configuration is bw=500Mbps, $delay$=25ms, per=10^{-6}, and $qsize$=0.5BDP. Web-like background traffic consumes about 100Mbps, and there are one legacy FTP flow and one VoIP flow. The number of HSCC flows (N) is set to 2, 6, 10, 16, and 20.



(a) Utilization Ratio of the Bottleneck Link



(b) Average Delay Experienced by VoIP Packets

Fig. 13. Scalability with Flow Number (Testbed Evaluation)

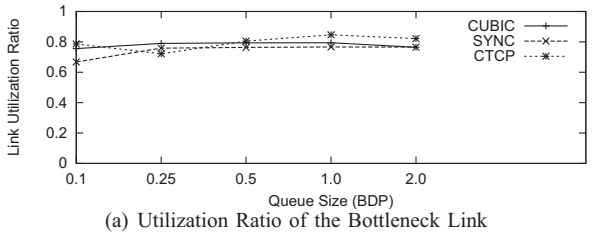
The duration of each experiment is 600 seconds. Figure 13 plots utilization ratio of the bottleneck link and user experience of VoIP. Link utilization observed is lower than 1 due to Iperf measurement, header overhead, and Dummynet inaccuracy when emulating high speed networks. Based on these plots, we find that when N is small (2 or 6), CTCP cannot fully utilize

the bottleneck link. A possible reason is that when per-flow throughput is high, CTCP is more likely to regard the noise in RTT samples as congestion signal. When N is large, CTCP can utilize the bottleneck link efficiently, but VoIP packets suffer longer delay too. Cubic TCP can efficiently utilize the bottleneck link in all cases. However, VoIP packets also suffer longer delay, and performance degrades when N increases. As for Sync-TCP, it can efficiently utilize the bottleneck link and keep the friendliness to cross traffic independent of the number of competing flows.

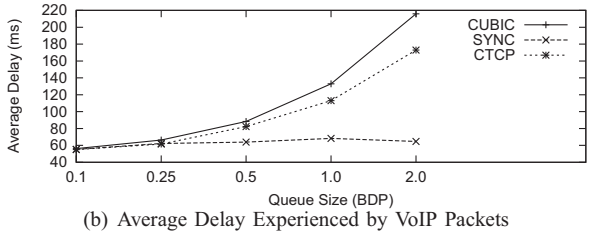
While the results are not shown here, we observed that when Cubic TCP is used, throughput of the legacy FTP flow drops by about 50%, compared to the case where either Sync-TCP or CTCP is used.

D. Effects of Buffer Sizes

In this set of experiments, we vary the amount of buffer available on the router. The bottleneck link is emulated as bw=1Gbps, $delay$=25ms, per=10^{-6}, and $qsize$ varies from 0.1BDP to 2.0BDP. 30 flows, which are driven by CTCP, Cubic TCP, or Sync-TCP, are established between SENDER and SINK. As for background traffic, web-like traffic consumes about 200Mbps and there are also a VoIP flow and 30 legacy FTP flows. Each experiment runs for 600 seconds.



(a) Utilization Ratio of the Bottleneck Link



(b) Average Delay Experienced by VoIP Packets

Fig. 14. Effects of Different Buffer Sizes (Testbed Evaluation)

Figure 14(a) plots the utilization ratio of the bottleneck link. It indicates that Sync-TCP will cause slightly lower utilization ratio when the queue size is smaller than the value expected by Sync-TCP ($bw * 12$ms). Please note that this value is independent of flow number.

Figure 14(b) plots the average delay experienced by VoIP packets, when queue size varies. It indicates that when queue size is large, the deployment of CTCP and Cubic TCP can severely degrade user experience of VoIP.

E. Dynamic Scenario

In this set of experiments, we evaluate whether Sync-TCP can work well with changes in HSCC flows and cross traffic. The bottleneck link is emulated as bw=1Gbps, $delay$=25ms, per=10^{-6}, and $qsize$=0.5BDP. Figure 15 shows the timing of traffic generation for this experiment.

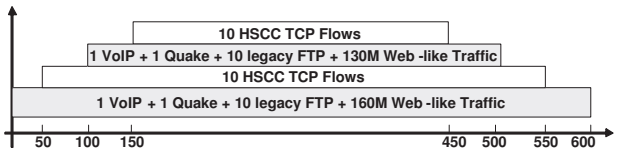


Fig. 15. Flow Arrive and Leave Sequence

Results from D-ITG and Iperf show that all three TCP variants can fully utilize the bottleneck link. Table I shows the VoIP user experience (delay and packet loss rate) when different HSCC TCP variants are adopted. It can be seen that Cubic TCP has the highest delay and loss rate, followed by CTCP and finally Sync-TCP. Another way to interpret the delay measurement is to translate this delay into normalized average buffer occupancy. For example, for Sync-TCP, normalized average buffer occupancy is $\frac{61ms-RTPD}{MaximumQueuingDelay}$ = 44%, where RTPD = 50ms and Maximum Queueing Delay is 25ms since queue size is set to 0.5BDP. This value can be more than 100% since there is imprecision in the scheduling interval and there can be buffering elsewhere. Interpreted this way, the impact of Cubic TCP and CTCP on VoIP traffic can be very significant if large buffer is used.

VoIP Packets	Cubic TCP	CTCP	Sync-TCP
Average Delay (ms)	86	75	61
Normalized Buffer Occupancy	144%	100%	44%
Packet Loss Rate	0.0075	0.0061	0.0031

TABLE I
VOIP USER EXPERIENCE

Figure 16 shows that Sync-TCP flows starting at different times can converge to the equilibrium point quickly, about 10s in this case. As for Cubic-TCP and CTCP, their results are much worse. In many cases, these HSCC flows cannot converge to a steady state within the duration of the experiment, 600 seconds.

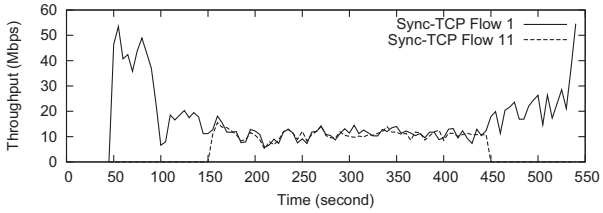


Fig. 16. Throughput Trajectory of Two Sync-TCP Flows

F. Summary of Testbed Evaluations

For each of the above described experiments, we calculate the tuple consisting of (1) the ratio of wasted bandwidth (1 - link utilization ratio) and (2) the normalized average buffer

occupancy. These tuples are plotted in figure 17, each point corresponding to a testbed experiment. For a HSCC algorithm, the ideal performance is that there is no wasted bandwidth and the buffering occupancy is minimum. Hence, the ideal performance is indicated by the position (0,0). Points closer to this location have better performance since they achieve good tradeoff between efficiency and friendliness.

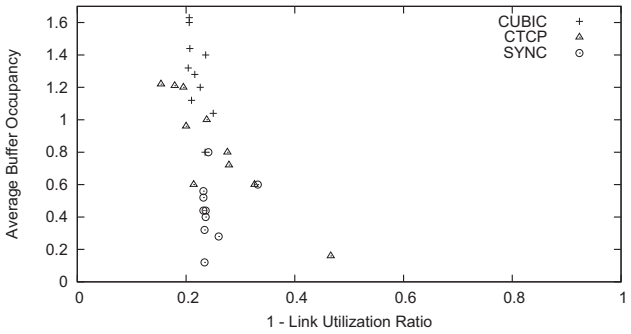


Fig. 17. Tradeoff Between Efficiency and Friendliness

From Figure 17, Cubic TCP always achieves very good efficiency. However, the cross traffic also always experience long queue delay and possible high packet loss rate. Hence, Cubic TCP achieves high efficiency at the cost of hurting cross traffic (web surfing, VoIP, etc.). As for CTCP, there is quite a lot of variation in efficiency. When CTCP performs good in efficiency, buffer occupancy also becomes high.

Figure 17 shows that Sync-TCP achieves the best tradeoff between efficiency and friendliness. Its performance is more reliable in terms of providing good link utilization while making sure that average delay is also low.

VIII. Conclusion

In this paper, Sync-TCP, a new delay-based high speed congestion control algorithm, is proposed for speeding up bandwidth-greedy and elastic applications on long fat network pipes of the Internet, while not hurting the cross traffic applications, especially the interactive ones. Sync-TCP is designed to drive these network pipes to operate around the knee and distribute the residual bandwidth fairly among competing flows even when the number of competing flows vary and their RTPDs differ significantly.

Extensive simulations and preliminary testbed evaluations indicate that Sync-TCP does achieve its design goals. In the next step, more testbed evaluations and live experiments in the Internet will be carried out for evaluating Sync-TCP further.

Acknowledgements

This work is supported in part by University Research Committee, National University of Singapore under grant T1 251RES0702. The authors would like to thank Prof. Thomas La Porta (the shepherd) and the anonymous reviewers for their insightful comments.

References

[1] S. Floyd, "Highspeed tcp for large congestion windows," RFC 3649, Dec. 2003.
[2] I. Rhee and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," in *PFLDnet Workshop*, 2005.
[3] D. Leith, R. Shorten, and Y. Lee, "H-tcp: A framework for congestion control in high-speed and long-distance networks," in *PFLDnet Workshop*, 2005.
[4] C. Jin, D. X. Wei, and S. H. Low, "Fast tcp: motivation, architecture, algorithms, performance," in *INFOCOM*, 2004.
[5] S. Liu, T. Basar, and R. Srikant, "Tcp-illinois: A loss and delay-based congestion control algorithm for high-speed networks," in *ValueTools*, 2006.
[6] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *INFOCOM*, 2006.
[7] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of tcp pacing," in *INFOCOM*, 2000.
[8] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *Computer Communication Review*, vol. 19, pp. 56–71, Oct. 1989.
[9] T. Kelly, "Scalable tcp: improving performance in highspeed wide area networks," *Computer Communication Review*, vol. 33, pp. 83–91, 2003.
[10] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long distance networks," in *INFOCOM*, 2004.
[11] D.Leith, R.Shorten, G.McCullagh, J.Heffner, L.Dunn, and F.Baker, "Delay-based aimd congestion control," in *PFLDnet Workshop*, 2007.
[12] A. Venkataramani, R. Kokku, and M. Dahlin, "Tcp nice: A mechanism for background transfers," in *OSDI*, 2002.
[13] A. Kuzmanovic and E. W. Knightly, "Tcp-lp: A distributed algorithm for low priority data transfer," in *INFOCOM*, 2003.
[14] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994.
[15] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," in *SIGCOMM*, 2002.
[16] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," in *SIGCOMM*, 2005.
[17] D. Leith, R.N.Shorten, and G.McCullagh, "Experimental evaluation of cubic-tcp," in *PFLDnet Workshop*, 2007.
[18] J. Cao, W. S. Cleveland, Y. Gao, K. Jeffay, F. D. Smith, and M. Weigle, "Stochastic models for generating synthetic http source traffic," in *INFOCOM*, 2004.
[19] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems archive*, vol. 7, June 1989.
[20] R. Shorten, F. Wirth, and D. Leith, "A positive systems model of tcp-like congestion control: asymptotic results," *IEEE/ACM Transaction on Networking*, vol. 14, June 2006.
[21] D. D. Clark, "Window and acknowledgement stratedgy in tcp," RFC 813, July 1982.
[22] J. Kulik, R. Coulter, D. Rockwell, and C. Partridge, "A simulation study of paced tcp," CR-2000-209416, NASA, Tech. Rep., Jan. 2000.
[23] X. Wu, "Safely speeding up bandwidth-greedy and elastic applications of the internet," Technical Report, School of Computing, National University of Singapore, 2009, available online at http://cir.nus.edu.sg/synctcp/.
[24] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between tcp reno and vegas for deployment of tcp vegas to the internet," in *ICNP*, 2000.
[25] D. X. Wei and P. Cao, "Ns-2 tcp-linux: An ns-2 tcp implementation with congestion control algorithms from linux," in *ACM ValueTools - Workshop of NS-2*, 2006.
[26] "Iperf," http://dast.nlanr.net/Projects/Iperf/.
[27] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.
[28] A. Botta, A. Dainotti, and A. Pescap, "Multi-protocol and multi-platform traffic generation and measurement," in *INFOCOM 2007 (DEMO)*, 2007.