Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks

Kun Tan Jingmin Song
Microsoft Research Asia
Beijing, China
{kuntan, jmsong}@microsoft.com

Qian Zhang
DCS, Hong Kong Univ. of Sci. & Tech
Hong Kong
qianzh@cs.ust.hk

Murari Sridharan
Microsoft Corporation
One Microsoft way, Redmond, WA, USA
muraris@microsoft.com

Abstract –
Many applications require fast data transfer over high speed and long distance networks. However, standard TCP fails to fully utilize the network capacity in high-speed and long distance networks due to its conservative congestion control (CC) algorithm. Some works have been proposed to improve the connection's throughput by adopting more aggressive loss-based CC algorithms, which may severely decrease the throughput of regular TCP flows sharing the network path. On the other hand, pure delay-based approaches may not work well if they compete with loss-based flows. In this paper, we propose a novel Compound TCP (CTCP) approach, which is a synergy of delay-based and loss-based approach. More specifically, we add a scalable delay-based component into the standard TCP Reno congestion avoidance algorithm (a.k.a., the loss-based component). The sending rate of CTCP is controlled by both components. This new delay-based component can rapidly increase sending rate when the network path is under utilized, but gracefully retreat in a busy network when a bottleneck queue is built. Augmented with this delay-based component, CTCP provides very good bandwidth scalability and at the same time achieves good TCP-fairness. We conduct extensive packet level simulations and test our CTCP implementation on the Windows platform over a production high-speed network link in the Microsoft intranet. Our simulation and experiments results verify the properties of CTCP.

I. INTRODUCTION

Moving bulk data quickly over high-speed data networks is a requirement for many applications. Currently, most of the applications use the Transmission Control Protocol (TCP) to transmit data over the Internet. TCP provides reliable data transmission with an embedded congestion control algorithm [1]. This effectively removes congestion collapses in the Internet by adjusting the sending rate according to the available bandwidth of the network. However, it has been reported that TCP substantially underutilizes network bandwidth over high-speed and long distance networks [2] due to its conservative congestion control behavior.

Recent research has proposed many approaches to address this issue. One class of approaches modifies the increase/decrease parameters of the TCP congestion avoidance algorithm (CAA) and makes it more aggressive. Like TCP, approaches in this category are loss-based that use packet loss as the only indication of congestion. Some typical proposals include HSTCP [2], STCP [3], and BIC-TCP [5]. Another class of approaches, by contrast, is delay-based, which deduce congestion based on the round trip time (RTT) RTT variations, e.g., FAST TCP [4], and will gracefully reduce the sending rate upon the increase of RTT. All aforementioned approaches are shown to overcome TCP's deficiencies in high bandwidth-delay networks to some extent. However, in this paper, we argue that besides the scalability property, a new high-speed protocol must be *TCP-friendly* as well in order to be deployed progressively in a production network. Here, we define the TCP-friendliness as that *the new protocol should not reduce the performance of other regular TCP (Reno) flows competing on the same path*. This means that the high-speed protocols should only make better use of free available bandwidth, but not steal bandwidth from other flows.

For existing loss-based high-speed solutions (e.g. [2][3][5]), it is essential to be highly aggressive to efficiently utilize the link capacity. However, this aggressiveness also causes severe unfairness to existing TCP (Reno) flows. This is because these aggressive high-speed flows will generate much more self-induced packet losses and therefore reduce the throughput of regular TCP flows. On the other hand, delay-based approaches, although they can achieve high efficiency and good RTT fairness in a network where the majority flows are delay-based, will suffer from significant low throughput if most competing flows are loss-based, e.g. TCP-Reno. The reason is that delay-based approaches reduce their sending rate when bottleneck queue builds up in order to avoid self-induced packet losses. However, this behavior will encourage loss-based flows to increase their sending rate further as they may observe less packet loss. As a consequence, the loss-based flows will obtain much more bandwidth than their share, while delay-based flows may even be starved.

In this paper, we propose a new congestion control protocol for high-speed and long delay networks, which is scalable but at the same time maintains a good TCP-friendly property. Our new protocol is a synergy of both delay-based and loss-based congestion avoidance approaches, which we name *Compound TCP* (CTCP). The key idea of CTCP is to add a scalable delay-based component into standard TCP. This delay-based component has a scalable window increasing rule that not only can efficiently probe the link capacity, but also reacts early to congestion by sensing the changes in RTT (i.e. if a bottleneck queue is sensed, the delay-based component gracefully reduces the sending rate.). This way, CTCP achieves good TCP fairness.

We conduct extensive packet level simulations using the NS 2 simulator [11]. We have also implemented CTCP on a Windows platform and made some preliminary experiments over a production high-speed backbone in the Microsoft intranet. The simulation and the experiments results suggest that CTCP is a promising algorithm to achieve both high link utilization and good TCP fairness.

The rest of paper is organized as follows. In the next section, we briefly review some existing approaches for high-speed networks. Then, we propose our design of CTCP in Section III. We present performance evaluation of CTCP using simulations in Section IV. CTCP implementation and experimental results on a productive network are presented in Section V. We conclude the paper in Section VI.

II. RELATED WORKS

The standard TCP congestion avoidance algorithm employs an *additive increase and multiplicative decrease* (AIMD) scheme, which is very conservative to increase the *congestion window* (cwnd) (one Maximum Segment Size (MSS) per RTT) and very aggressive to decrease the window by half upon a packet loss. For a high-speed and long delay network, it will take standard TCP an unreasonably long time to recover the sending rate after a single loss event. Moreover, it is well-known now that the average TCP *congestion window* is inversely proportional to the square root of the packet loss rate. Therefore, it requires an extremely small packet loss rate to sustain a large window, which may not be practical in real networks.

One straightforward way to overcome this limitation is to modify TCP's increase/decrease control rule in its congestion avoidance stage. More specifically, in the absence of packet loss, the sender increases cwnd more quickly and decreases it more gently upon a packet loss.

STCP [3] alters TCP's AIMD congestion avoidance scheme to MIMD (*multiplicative increase and multiplicative decrease*). Specifically, STCP increases cwnd by 0.01 MSS on every received ACK and reduces cwnd to its 0.875 times upon a packet loss. HSTCP [2], on the other hand, still mimics the AIMD scheme, but with varying increase/decrease parameters. As cwnd increases from 38 packets to 83,333 packets, the decrease parameter reduces from 0.5 to 0.1, while the increase parameter increases accordingly. HSTCP is less aggressive than STCP, but is far more aggressive than the standard TCP. In [5], BIC-TCP is proposed to use a binary increase scheme and switching to AIMD with constant parameters when cwnd is large. The BIC's growth function can still be too aggressive for TCP, especially under short RTT or low speed networks.

In a mixed network environment, the aggressive behavior of the above approaches may severely degrade the performance of regular TCP flows whenever the network path is already highly utilized. When an aggressive high-speed variant flow traverses the bottleneck link with other standard TCP flows, it may increase its own share of bandwidth by reducing the throughput of other competing TCP flows, as the aggressive high-speed variants will cause much more self-induced packet losses on bottleneck links, and push back the throughput of the regular TCP flows.

Another class of high speed protocols, like FAST TCP [4], chooses to design a new congestion control scheme which takes RTT variances as a congestion indicator. These delay-based approaches are more-or-less derived from seminal work of TCP Vegas [7]. One core idea of delay-based congestion avoidance is that the increase of RTT is considered as early indicator of congestion, and the sending rate is reduced to avoid self-induced buffer overflow. In this way, they will not cause large queueing delay and reduce packet losses. FAST TCP can be viewed as a scalable version of TCP Vegas. FAST TCP incorporates multiplicative increase if the buffer occupied by the connection at the bottleneck is far less than some pre-defined threshold α, and switches to linear increase if it is near α. Then, FAST tries to maintain the buffer occupancy around α and reduces the sending rate if the delay is further increased. However, previous work reveals that delay-based approaches may not be able to obtain fair share when they are competing with loss-based approaches like standard TCP [12][13]. This can be explained as follows. Consider a delay-based flow, e.g. Vegas or FAST, shares a bottleneck link with a standard TCP. Since the delay-based flow tries to maintain a small number of packets in the bottleneck queue, it will stop increasing its sending rate when the delay reaches some value. However, the loss-based flow will not react to the increase of delay, and continues to increase the sending rate. This, observed by the delay-based flow, is considered as congestion indication and therefore the sending rate of the delay-based flow is further reduced. In this way, the delay-based flow may obtain far less bandwidth than its fair share.

III. THE COMPOUND TCP

The design of *Compound TCP* is to satisfy to efficiency requirement and TCP friendliness requirement simultaneously. The key idea is that if the link is under-utilized, the high-speed protocol should be aggressive and increase the sending rate quickly. However, once the link is fully utilized, being aggressive is no longer good, as it will only cause problems like TCP unfairness. We note that delay-based approaches already have this nice property of adjusting its aggressiveness based on the link utilization, which is observed by the end-systems from the increase in the packet delay. However, as mentioned in the previous section, the major weakness of delay-based approaches is that they are not competitive to loss-based approaches. And this weakness is difficult to be remedied by delay-based approaches themselves.

Having made this observation, we propose to adopt a synergic way that combines a loss-based approach with a delay-based approach for high speed congestion control. For easy understanding, let's imagine application *A* communicates with application *B* simultaneously using two flows. One is a standard loss-based TCP flow, and the other is a delay-based flow. When the network is underutilized, *A* can get an aggregated communication throughput, with *B*, which is the sum of both flows. With the increase of the sending rate, a queue is built at the bottleneck, and the delay-based flow gradually reduces its sending rate. The aggregated throughput for the communication may gradually reduce but is bound by the standard TCP flow.

Then, there comes the core idea of our novel CTCP, which incorporates a scalable delay-based component into the standard TCP congestion avoidance algorithm. This scalable delay-based component has a rapid window increase rule when the network is sensed to be under-utilized and gracefully reduces the sending rate once the bottleneck queue is built. With this delay-based component as an auto-tuning knob, *Compound TCP* is scalable yet TCP-friendly:

1) CTCP can efficiently use the network resource and achieve high link utilization. In theory, CTCP can be very fast to obtain free network bandwidth, by adopting a rapid increase rule in the delay-based component, e.g. multiplicative increase. However, in this paper, we choose CTCP to have similar aggressiveness to obtain available bandwidth as HSTCP.

2) CTCP has good TCP-fairness. By employing the delay-based component, CTCP can gracefully reduce the sending rate when the link is fully utilized. In this way, a CTCP flow will not cause more self-induced packet losses than a standard TCP flow, and therefore maintains fairness to other competing regular TCP flows.

Note that CTCP also has improved RTT fairness compared to regular TCP. This is due to the delay-based component employed in the CTCP congestion avoidance algorithm [9]. It is known that delay-based flow, e.g. Vegas, has better RTT fairness than the standard TCP.

A. Architecture

As explained earlier, CTCP is a synergy of a delay-based approach with a loss-based approach that implements a new scalable delay-based component within the standard TCP congestion avoidance algorithm (called *loss-based component*). To do so, a new state variable is introduced, namely, *dwnd* (Delay Window), which controls this delay-based component in CTCP. The conventional congestion window, *cwnd*, remains untouched, which controls the loss-based component in CTCP. Then, the CTCP sending window now is controlled by both *cwnd* and *dwnd*. Specifically, the TCP sending window (called *window* hereafter) is now calculated as follows:

$$win = \min(cwnd + dwnd, awnd)\ , \qquad (1)$$

where *awnd* is the advertised window from the receiver.

The update of *dwnd* will be elaborated in detail in next subsection, while the update of *cwnd* is in the same way as in the regular TCP in the congestion avoidance phase, i.e., *cwnd* is increased by one MSS every RTT and halved upon a packet loss event. Therefore, the increment of *cwnd* on arrival of an ACK is:

$$cwnd = cwnd + 1/win\ , \qquad (2)$$

where *win* is defined with equation (1).

Slow-Start behavior of regular TCP at the start-up of a new connection does not change in CTCP. This is since we believe slow-start, which exponentially increases the window, is quick enough even for many fast and long distance environments we target [2]. We initially set *dwnd* to zero if the connection is in slow-start state, and the delay-based component is effective only when the connection is working at congestion avoidance phase.

B. Design of delay-based congestion avoidance

We design the delay-based congestion avoidance algorithm to have the following properties. Firstly, it should have an aggressive, scalable increase rule when the network is sensed to be under-utilized. Secondly, it should also reduce the sending rate accordingly when the network is sensed to be fully utilized. By reducing its sending rate, the delay-based component yields ways for competing TCP flows to ensure TCP fairness of CTCP. Lastly, it should also react to packet losses. It is because packet losses may still be an indicator of heavy congestion, and hence reducing sending rate upon packet loss is a necessary conservative behavior to avoid congestion collapse.

Our algorithm for delay-based component is derived from TCP Vegas. A state variable, called *baseRTT*, is maintained as an estimation of the transmission delay of a packet over the network path. When the connection is started, *baseRTT* is updated by the minimal RTT that has been observed so far. An exponentially smoothed current RTT, *sRTT*, is also maintained. Note that both *baseRTT* and *sRTT* should be of high resolution. Then, the number of backlogged packets of the connection can be estimated by following algorithm:

$$\begin{aligned} Expected &= win / baseRTT \\ Actual &= win / RTT \\ Diff &= (Expected - Actual) \cdot baseRTT \end{aligned}$$

The *Expected* gives the estimation of throughput we get if we do not overrun the network path. The *Actual* stands for the throughput we really get. Then, *(Expected – Actual)* is the difference between the expected throughput and the actual throughput. When multiplying by *baseRTT*, it stands for the amount of data that injected into the network in last round but does not pass through the network in this round, i.e. the amount of data backlogged in the bottleneck queue. An early congestion is detected if the number of packets in the queue is larger than a threshold γ. If $diff < \gamma$, the network path is determined as under-utilized; otherwise, the network path is considered as busy and delay-based component should gracefully reduce its window.

The increase law of the delay-based component should make CTCP more scalable in high-speed and long delay pipes. In this paper, we choose the CTCP window evolution to have the binomial behavior. More specifically, when no congestion occurs (i.e. sensing neither queuing delay nor packet losses), the CTCP window increases as follows

$$win(t+1) = win(t) + \alpha \cdot win(t)^k\ . \qquad (3)$$

If there is a loss, the window is multiplicatively decreased,

$$win(t+1) = win(t) \cdot (1-\beta)\ . \qquad (4)$$

Parameters of α, β and k are tunable to give out desirable scalability, smoothness and responsiveness, which will be discussed in detail in Section III.C.

Considering there is already a loss-based component in CTCP, the delay-based component needs to be designed to only fill the gap, and the overall CTCP should follow the behavior defined in (3) and (4). We summarize the algorithm for the delay-based component as in (5)

$$dwnd(t+1) = \begin{cases} dwnd(t) + (\alpha \cdot win(t)^k - 1)^+, \text{if } diff < \gamma \\ \left(dwnd(t) - \zeta \cdot diff\right)^+, \text{if } diff \geq \gamma \\ \left(win(t) \cdot (1-\beta) - cwnd/2\right)^+, \text{if loss is detected} \end{cases}, \quad (5)$$

where $(.)^+$ is defined as max (., 0). The first line shows that in the increase phase, *dwnd* only needs to increase $(\alpha \cdot win(t)^k - 1)^+$ packets, since the loss-based component (*cwnd*) will also increase by 1 packet. Similarly, when there is a loss, *dwnd* is set to the difference between the desired reduced window size and that can be provided by *cwnd*. The rule on the second line is important. It shows that *dwnd* does decrease when the queue is built, and this is the core for CTCP to preserve good TCP fairness. Here, ζ is a parameter that defines how rapidly the delay-based component should reduce this window when early congestion is detected. Note that *dwnd* will never be negative. Therefore, CTCP window is low-bounded by its loss-based component (i.e. a standard TCP).

In the above control laws, we assume the loss is detected by three duplicate ACKs. If a retransmission timeout occurs, *dwnd* should be reset to zero and the delay-based component is disabled. This is since after a timeout, the TCP sender is put into the slow-start state. After the CTCP sender exits the slow-start recovery state, the delay-based component may be enabled once more. Also, following the common practice of high-speed protocols, CTCP also reverts to standard TCP behavior when the window is small. Delay-based component only kicks in when *win* is larger than some threshold, W_{low}.

C. Parameter Setting

Our analysis [9] on CTCP has shown that if the network path is underutilized and the queue is neglectable, the response function of CTCP is

$$\Lambda = \frac{1}{R \cdot \alpha^{\frac{1}{2-k}} \cdot \left(1-(1-\beta)^{1-k}\right)} \left[\frac{1-(1-\beta)^{2-k}}{2-k}\right]^{\frac{1-k}{2-k}} \cdot \frac{1}{p^{\frac{1}{2-k}}}, \quad (6)$$

where *p* is the packet loss rate and *R* is the round trip time.

We intend to choose the CTCP parameters so it has similar aggressiveness to HSTCP when the network is underutilized. Therefore, by comparing equation (6) with the response function of HSTCP, we get $k = 0.8$; $\alpha = 1/8$; and $\beta = 1/2$. Note that, it is rather difficult to implement an arbitrary power calculation using integer algorithm. Therefore, we choose *k* equal to 0.75, which can be implemented with a fast integer algorithm of square root.

The threshold γ is also an important parameter. From the control law (5), we can see that it requires the connection to have at least γ packets backlogged in the bottleneck queue to detect early congestion. In previous work [9], we empirically choose γ to be a fixed value. In this paper, we propose an adaptive way that dynamically adjusts γ based on the network configuration, which is named as *gamma auto-tuning*.

Consider a simple network model in Figure 1. Assume the bottleneck router buffer can contain *B* packets and there are *m* concurrent flows. So, the condition that the early detection of CTCP is less effective is: a) there are less than γ packets buffer allocated for each flow, or

$$\gamma > \frac{B}{m}; \quad (7)$$

and b) the delay-based component does kick in. This is equivalent to the expected window of each flow is larger than W_{low},

$$W_{low} < \frac{B + uT}{m}, \quad (8)$$

where uT is the bandwidth delay production (BDP) of the link.

We further assume that the buffer deployed in the bottleneck is a fraction of whole BDP, or $B = \kappa uT$. Simplifying equation (7) and (8), we get

$$\gamma > W_{low} \cdot \frac{\kappa}{1+\kappa}. \quad (9)$$

Therefore, in order to avoid this case, we should have

$$\gamma = \max(\gamma_{\min}, W_{low} \cdot \frac{\kappa}{1+\kappa}). \quad (10)$$

where parameter κ can be estimated using

$$\kappa = \frac{R_{\max} - R_{\min}}{R_{\min}}, \quad (11)$$

where R_{max} is the maximal RTT while R_{min} is the maximal RTT observed so far.

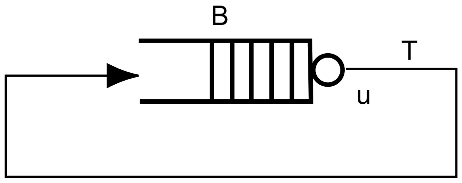

Figure 1. A simple network model.

IV. PERFORMANCE EVALUATION

In this section, we present some performance results regarding the efficiency and TCP-friendliness of CTCP. We use NS-2 to conduct simulations on a typical dumbbell topology. The bottleneck link capacity is set to 1Gbps unless otherwise pointed out. Each simulation run lasts at least 150s. The parameter $\gamma_{\min}$ is set to 3 packets and W_{low} is 41 packets.

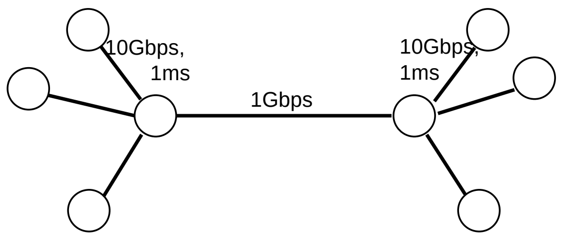

Figure 2. The dumbbell topology. The bottleneck link capacity is set to 1Gbps.

A. *Throughput under random packet loss*

The first simulation is to verify the ability that CTCP can effectively utilize the network link capacity. We set the RTT to 100ms in this simulation and bottleneck buffer size is 1500 packets. We artificially add random losses on the bottleneck link, and the packet loss rate varies from 10^{-2} to 10^{-6}. For each loss rate, we test 4 flows of CTCP, HSTCP, and regular TCP, respectively. Figure 3 shows the throughput for each type of TCP variant. We see that regular TCP can not scale well with increased packet loss rate. It can only use up to 73% of the link capacity even with a very low packet loss rate 10^{-6}. CTCP and HSTCP have much higher link utilization, and the link is fully utilized when the loss rate is less than 10^{-6}. CTCP has a bit higher throughput than HSTCP. This is since the delay-based component in CTCP introduces much less self-induced loss compared to HSTCP.

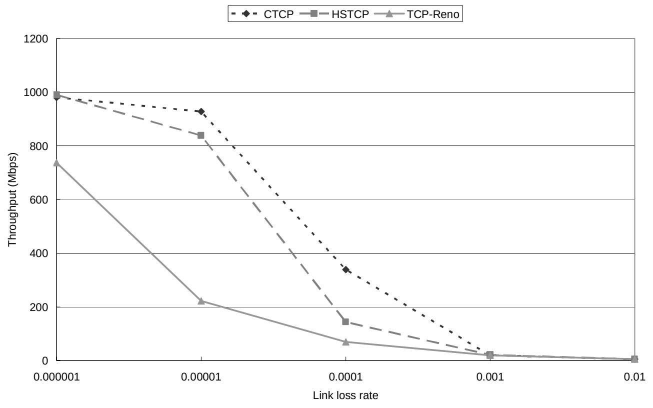

Figure 3. Aggregated throughput under different link packet loss rate. The X-axis shows the artificially introduced random loss rates on the bottleneck link.

B. *TCP fairness under random packet loss*

To quantify the throughput reduction of the regular TCP flows due to the introduction of CTCP flows, we define a new metric to measure the TCP fairness, named *bandwidth stolen*.

Definition 1: bandwidth stolen. Let P be the aggregated throughput of m regular TCP flows when they compete with another l regular TCP flows. Let Q be the aggregated throughput of m regular TCP flows when they compete with another l high-speed protocol flows in the same network topology. Then, $B_{stolen} = \frac{P-Q}{P}$ is the *bandwidth stolen* by high-speed protocol flows from regular TCP flows.

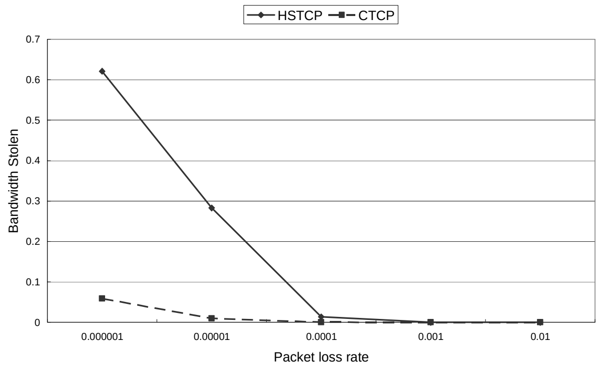

Figure 4. Bandwidth stolen under different link packet loss rate. The x-axis shows the artificially introduced random loss rates on the bottleneck link.

To calculate the bandwidth stolen, we first run eight regular TCP flows and record the average throughput as the baseline. Then, we run four high-speed flows against four regular TCP flows. We compare the average throughput of the regular TCP with the baseline. Figure 4 shows the bandwidth stolen of HSTCP and CTCP under different random link packet loss rates. The network setup is the same as previous simulation in Section IV.A. We can see that with the decrease of link packet loss rate, HSTCP becomes more aggressive and strongly pushes regular TCP back to gain more bandwidth. It is clearly shown in the figure that regular TCP flows have up to 63% throughput reduction. However, CTCP is much fairer to regular TCP and causes less than 10% throughput reduction.

C. *TCP fairness with various link speed*

In this simulation, we vary the bottleneck bandwidth from 20Mbps to 1Gbps, and set RTT to 100ms. The bottleneck buffers size is set to the BDP of the path. We run four high-speed flows against four regular TCP flows. Figure 5 shows the bandwidth stolen of HSTCP and CTCP. With the increase of bottleneck bandwidth, HSTCP becomes more and more unfair to regular TCP as it becomes more aggressive. It causes the regular TCP flow to have up to 80% throughput reduction. In all tested cases, CTCP maintains TCP fairness very well.

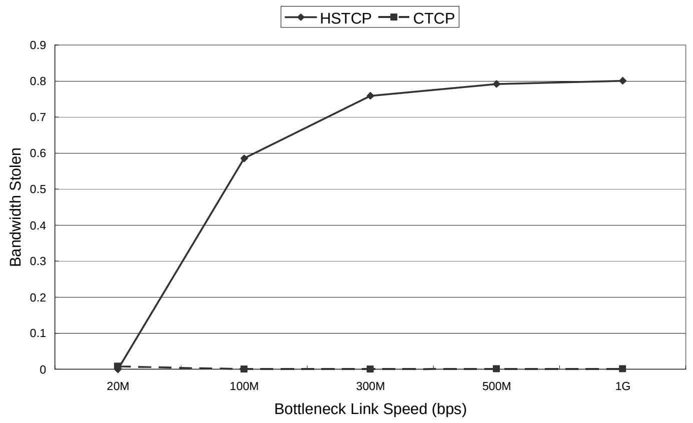

Figure 5. Bandwidth stolen under different bottleneck link speed.

D. TCP fairness with various bottleneck buffer size

In this simulation, we maintain the bottleneck speed as 1Gbps and RTT as 100ms, and vary the bottleneck buffer size from 300 packets (4% of BDP) to 6000 packets (72% of BDP). We run four high-speed flows against four regular TCP flows. Figure 6 shows the bandwidth stolen of HSTCP as well as CTCP. It shows that HSTCP constantly causes severe throughput reduction of regular TCP flows and therefore unfair to regular TCP. CTCP, however, preserves excellent TCP fairness in the tested buffer range. Note that due to the adaptive γ setting, CTCP can work well even if the bottleneck buffer is very small, e.g. 4% of BDP.

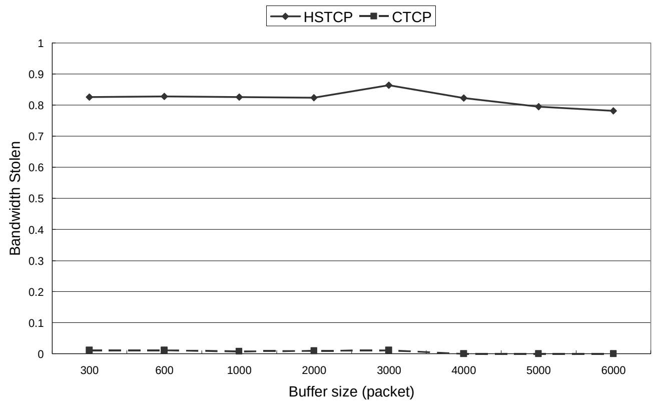

Figure 6. Bandwidth stolen with different buffer size at bottleneck router.

E. Reverse traffic

As the primary design goal of CTCP is to improve the connection's throughput without reducing the performance of competing regular TCP flows, we choose to use the increase of round trip time as an indication of early congestion. However, there are some concerns on the impacts of reverse traffic on CTCP, as the reverse traffic will generally add delay in the ACK path and therefore enlarge the RTT. In [14], Cottrell *et al* finds delay-based protocol, e.g. FAST, is "very handicapped by reverse traffic". In this section, we also evaluate the performance of CTCP under similar situations.

In our simulation, the bottleneck bandwidth is 1Gbps, and the round trip delay is 30ms. We set the bottleneck buffer on both forward and the reverse path to be 750 packets. We run one forward flow (CTCP, HSTCP and regular TCP, respectively) with different number of reverse regular TCP flows. Table 1 summarizes the results. The first column shows the number of reverse flows. For each forward flow type, namely HSTCP, CTCP and regular TCP, column FW shows the throughput of forward flow; column R shows the aggregated throughput of reverse flow; and column Sum shows the summation of all flows.

We can see that CTCP constantly has lower throughput than HSTCP. This reflects that the reverse traffic has impact on CTCP throughput as it creates a queue on the reverse path that delays ACKs. However, CTCP still has improved throughput compared to regular TCP. This is reasonable since CTCP is lower bounded by its loss-based component, and the delay-based component of CTCP reacts to the increase of RTT by reducing only its *dwnd*.

Another observation we have from the simulation is that the forward flow also impacts the reverse flows. This is shown by the aggregated throughput of reverse flows. The aggregated throughput of reverse flows with HSTCP is constantly less than that with CTCP or Regular TCP. For example, if the forward flow is HSTCP, the aggregated throughput of two reverse flows is 430Mbps, while this number is 662Mbps and 664Mbps, respectively, if the forward flow is CTCP and regular TCP. Moreover, the summations of throughput of all flows including both forward and reverse traffic are, surprisingly, very similar in all tested cases (as shown in Column Sum). This suggests to us that aggressive behavior in the forward path, for HSTCP, "trades" some throughput of the reverse traffic to the forward throughput. We hypothesize this is because the aggressive forward flow will generate more ACKs on the reverse path, which are contending for the buffer with the reverse data packets[1] and may create more packet losses in the reverse path. We plan to investigate this problem further in our future work.

The queue on the reverse path suggests there is congestion on the reverse path. Whether or not to react to the reverse congestion is under debate. However, based on the conservative design goal of CTCP, we believe reacting to both forward and backward queue is a reasonable choice to be friendly to (both forward and backward) regular TCP flows.

Table 1. Throughput under reverse traffic. Column RF# presents the number of reverse flows. Column FW presents the throughput of forward flow. Column R presents the aggregated throughput of reverse flows. Column Sum presents the sum of throughput including both forward and backward flows. The unit of data is in Mbps.

RF #	HSTCP			CTCP			Regular TCP		
	FW	R	Sum	FW	R	Sum	FW	R	Sum
1	818	338	1156	557	496	1053	491	531	1022
2	705	430	1136	397	662	1059	357	664	1021
4	653	442	1096	307	842	1133	291	827	1134
8	648	437	1085	272	850	1121	243	876	1119
16	480	619	1099	300	898	1198	271	900	1170

V. Experiment on a Production Network Link

We have implemented CTCP on the Microsoft Windows Platform by modifying the TCP/IP stack and we conduct some preliminary experiments over a production network link from Tukwila, WA, to the San Francisco Bay Area, California. The link is a part of the high-speed backbone of the Microsoft intranet. The link capacity is 1Gbps and the round trip delay is

[1] Many current routers allocate fixed memory slot for every packet. Therefore, a small packet, like ACK, may occupy the same memory slot as a full sized data packet.

around 30ms. There is some light-loading of cross-traffic on this link. Since we can not control the traffic on the network, the data presented below is the average of at least 5 runs for experiments.

A. *Throughput*

We first test the throughput of CTCP over that link. We separately test regular TCP and CTCP with one, two and three concurrent flows. Figure 7 shows the throughput results of the experiments. We can see CTCP generally improves the throughput by 28% to 52% compared to regular TCP.

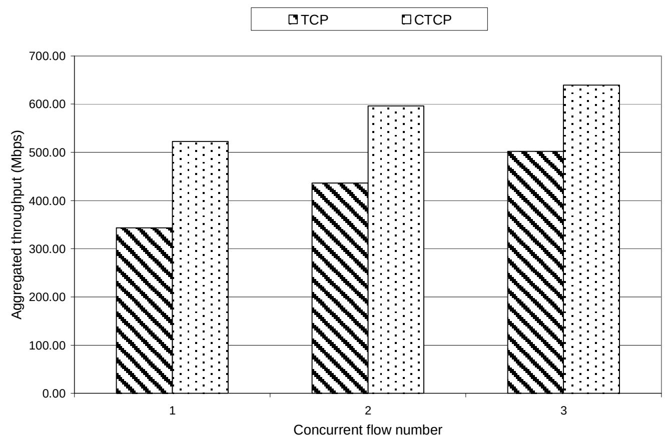

Figure 7. The aggregated throughput. The x-axis shows the number of concurrent flows.

B. *TCP friendliness*

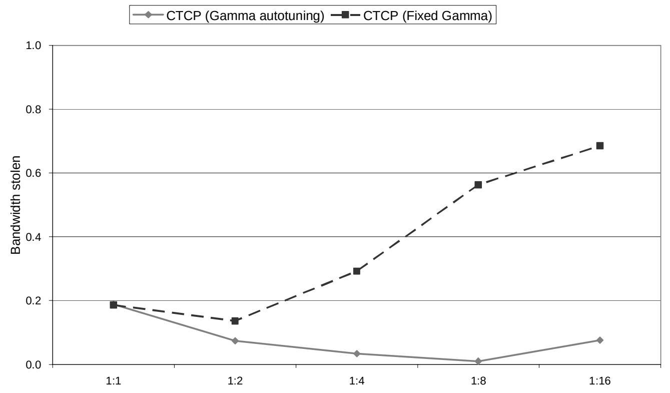

Figure 8. The bandwidth stolen of CTCP with and without gamma auto-tuning. The markers on x-axis, 1:n, mean that there are one regular flow and n CTCP flows.

We conduct the following experiment to validate the TCP friendliness property of our CTCP implementation. We run one regular TCP flow simultaneously with several CTCP flows. Then, we measure the bandwidth reduction of the regular TCP flow with the increasing number of CTCP flows. Figure 8 shows the bandwidth stolen of CTCP with and without automatic gamma tuning. The dotted line presents the bandwidth stolen of CTCP with fixed gamma, as proposed in [9]. With the increase of concurrent CTCP flows, CTCP becomes more unfair to regular TCP. This is since the buffer provisioned on the network link is only about 10% of its BDP. In this case, the pre-configured γ=30 is much too high. With only a few concurrent flows, the average buffer size occupied by a flow is less than γ, and therefore the delay-based component in CTCP does not work well.

However, with gamma auto-tuning, CTCP automatically tunes a smaller γ based on the size of buffer provisioned on the link. Therefore, CTCP keeps good TCP friendliness even with more concurrent flows. Note that if there are a large number of flows sharing the bottleneck, the average size of a buffer occupied by a flow will eventually be less than γ, even with auto-tuning. However, in this case, the aggressiveness of CTCP is also reduced, since the average window size of a flow is much smaller. As a consequence, it still keeps good TCP-friendliness.

C. *Reverse traffic*

We also repeat the experiments with reverse traffic on the production network link. The results are presented in Figure 9. It shows that CTCP does improve the performance of TCP even with the presence of many reverse flows. These results are consistent with the simulation results presented in Section IV.E.

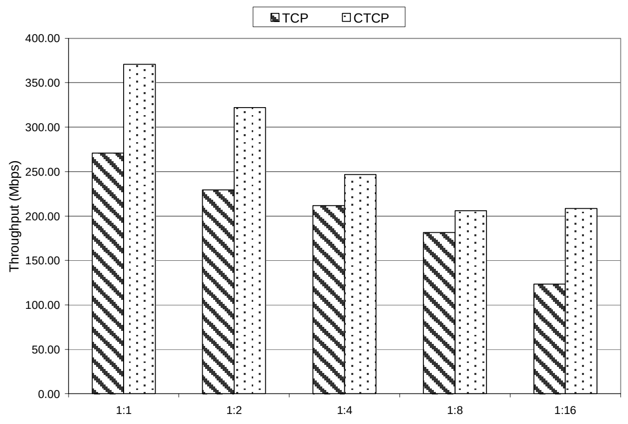

Figure 9. The throughput of the forward flow (CTCP and regular TCP, respectively) with reverse traffic. The markers on x-axis, 1:n, mean that there are one forward flow and n reverse regular TCP flows.

VI. CONCLUSIONS

In this paper, we present a novel congestion control algorithm for high-speed and long delay networks. Our Compound TCP approach combines a scalable delay-based component with a standard TCP loss-based component. The delay-based component can efficiently use free bandwidth with its scalable increasing law. When the network is congested, the delay-based component will gracefully reduce the sending rate, but the loss-based component keeps the throughput of CTCP lower bounded by TCP Reno. This way, CTCP will not be timid, nor induce more self-induced packet losses than a single TCP Reno flow, and therefore achieves good TCP fairness. We conducted extensive packet level simulations to evaluate the performance of CTCP. We also implemented CTCP on the Windows platform and conducted preliminary tests over a production high-speed network link in the Microsoft intranet. Our simulation and experiments results verify that CTCP can effectively utilize the link capacity, while at the same time maintaining excellent TCP fairness.

VII. ACKNOWLEDGEMENTS

The authors are grateful to Les Cottrell for his constructive comments and generous help for improving the presentation of this paper. The authors also would like to thank the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

[1] M. Allman, V. Paxson and W. Stevens, “TCP Congestion Control”, *RFC 2581*, April 1999.

[2] S. Floyd, “HighSpeed TCP for Large Congestion Windows”, *RFC 3649*, December 2003.

[3] Tom Kelly, “Scalable TCP: Improving Performance in HighSpeed Wide Area Networks”, *in First International Workshop on Protocols for Fast Long Distance Networks*, Geneva, February 2003.

[4] C. Jin, D. Wei and S. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance”, *In Proc IEEE Infocom 2004*.

[5] L. Xu, K. Harfoush and I. Rhee, “Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks”, *In Proc. IEEE InfoCOM 2004*.

[6] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, “Data management and transfer in high performance computational grid environments”, *Parallel Computing*, May 2002.

[7] L. Brakmo, S. O'Malley, and L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance”, *in Proceedings of the SIGCOMM '94 Symposium*, Aug. 1994.

[8] D. Bansal and H. Balakrishnan, “Binomial Congestion Control Algorithms”, i*n Proc Infocom 2001*.

[9] Kun Tan, Jingmin Song, Qian Zhang, “A Compound TCP Approach for Fast Long Distance Networks”, Microsoft Technical Report, 2005.

[10] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet”, *IEEE/ACM Trans. on Networking*, August 1999.

[11] The Network Simulation - NS2. Available at http://www.isi.edu/nsnam/ns/

[12] T. Bonald, “Comparison of TCP Reno and TCP Vegas via fluid approximation”, *Performance Evaulation*, 36(37):307-332, 1999.

[13] J. Mo, R.J. La, V. Anantharam, and J.Walrand, “Analysis and Comparison of TCP Reno and Vegas”, *in Proceedings of INFOCOM ’99*, March 1999.

[14] R. L. Cottrell, H. Bullot, and R. Hughes-Jones, “Evaluation of Advanced TCP stacks on Fast Long-Distance production Networks”, in *workshop of Protocols for Fast Long Distance networks*, 2004.