

A Simulation Study of Compound TCP

Xiuchao Wu

School of Computing, National University of Singapore
Computing 1, Law Link, Singapore 117590

Email:wuxiucha@comp.nus.edu.sg

July 14, 2008

Abstract—Compound TCP, an innovative delay-based congestion control algorithm, has been implemented in Windows Vista of Microsoft for utilizing network pipes with huge bandwidth delay product while avoiding to starve TCP flows and avoiding to be starved by TCP flows. Hence, Compound TCP may be widely used in the Internet and it is very important to evaluate its performance under many different network environments. This paper is such an attempt to systematically study Compound TCP by simulation. Through this study, we find that Compound TCP still has several shortcomings and it should be necessary to revise Compound TCP before deployment.

I. INTRODUCTION

Compound-TCP (CTCP) [1], an innovative delay-based high speed congestion control algorithm that can well utilize network pipes with huge BDP (bandwidth delay product) while avoiding to starve TCP flows and avoiding to be starved by TCP flows, has been proposed recently. Due to these good properties, CTCP has been implemented in Windows Vista of Microsoft. Hence, CTCP may be widely used in the heterogeneous Internet and it should be necessary to evaluate its performance under many different network environments. In this paper, we carried out such an evaluation by simulation. During this study, we mainly find the following shortcomings, and the possible reasons are also analyzed in the paper.

- 1) In many cases, CTCP can not detect congestion by queue delay and it only achieves binomial growth of its sending window. Hence, CTCP flows can not drive network to work at the knee [2], link utilization ratio is low, many segments are dropped within one congestion event, RTT fairness becomes bad, and cross TCP flows are adversely affected. In this situation, CTCP is worse than many loss-based high speed congestion control algorithms, such as HS-TCP [3] and BIC-TCP [4].
- 2) Competing CTCP flows show complex convergence behaviors and the fairness among them is unpredictable. When BDP of a network path is huge, convergence speed can be very slow and CTCP only achieves theoretically fairness since many flows die before they acquire their fair share of bandwidth.
- 3) CTCP still has several problems owned by TCP Vegas [5], such as re-routing and persistent congestion [6].

This paper is organized as follows. CTCP is first introduced in section II. Section III then describes the existing evaluation work and the methodology used by us for studying CTCP. Simulation results are presented and analyzed in section IV. The paper is concluded in section V.

II. COMPOUND TCP

It has been well known that TCP's congestion control algorithm [7] can not work well on network pipes with huge BDP [3], and many HSCC (high speed congestion control) algorithms have been proposed for utilizing these network pipes. In recent years, designing HSCC algorithm with the assistance of queue delay has become a hot and promising research topic due to the following reasons. First, loss-based HSCC algorithms can not solve the fundamental confliction between efficiency and TCP friendliness. Second, routers of the Internet are normally provisioned with large queue. Hence, there are detectable queue delays. Third, on network pipes with huge BDP, end-hosts can send more packets per RTT, get more RTT samples, and have a more accurate estimation of queue delay. Many delay-based HSCC algorithms, such as Fast-TCP [8] and TCP-Fusion [9], has been proposed, and Compound TCP (CTCP) [1] is the innovative delay-based HSCC algorithm proposed by Microsoft.

CTCP is a sender-centric proposal and only data sender needs to be changed. CTCP sender maintains two variables, $dwnd$ and $cwnd$, whose sum represents the sending rate probed by CTCP sender through congestion control. The real sending rate is determined by the sending window, $win = \min(cwnd + dwnd, awnd)$. Here, $awnd$ is the advertisement window from the receiver and is normally assumed to be larger than $cwnd + dwnd$ when studying congestion control. In this paper, unless specified, win directly refers to $cwnd + dwnd$.

In congestion avoidance phase, $cwnd$ is updated in the same way as in the regular TCP (AIMD). That means $cwnd$ is increased by one segment per RTT and is halved when segment loss is detected. As for the delay-based component of CTCP, $dwnd$ is updated based on queue delay. At the end of one RTT, CTCP sender estimates Δ (the number of packets backlogged in the queue of the bottleneck link) according to the method in equation 1, which has also been adopted by TCP Vegas.

$$\begin{aligned} \Delta &= (Th_{expected} - Th_{actual}) * baseRTT \\ &= \left(\frac{win}{baseRTT} - \frac{win}{RTT} \right) * baseRTT \end{aligned} \quad (1)$$

	$cwnd$	$dwnd$
Arrival of ACK	$cwnd = cwnd + \frac{1}{win}$	
End of RTT, $\Delta < \gamma$		$dwnd = dwnd + (\alpha * win^k - 1)^+$
End of RTT, $\Delta \geq \gamma$		$dwnd = (dwnd - \zeta * \Delta)^+$
Segment Loss is detected	$cwnd = \frac{cwnd}{2}$	$dwnd = (win * (1 - \beta) - \frac{cwnd}{2})^+$

TABLE I
CONGESTION CONTROL ALGORITHM OF CTCP

Table I shows the rules used by CTCP for updating $cwnd$ and $dwnd$, where $(.)^+$ is defined as $max(., 0)$. With these rules, when network is under-utilized ($\Delta < \gamma$) and there is no segment loss, CTCP sending window is increased binomially ($win = win + \alpha * win^k$ per RTT). Hence, CTCP can probe network bandwidth quickly. After that, although $cwnd$ is continuously increased, delay-based component of CTCP ($dwnd$) tries to maintain γ packets at the bottleneck link and drive network to work at the knee by reducing itself. Since each flow tries to maintain the same number of packets at the bottleneck, CTCP owns good RTT fairness. During this period, concurrent loss-based TCP flows could take bandwidth away from CTCP flows. Hence, TCP flows will not be adversely affected a lot. In addition, since $dwnd$ will never be less than zero, CTCP flow will never acquire less bandwidth than the bandwidth acquired when it uses TCP AIMD congestion control. That means CTCP solves the Achilles' heel of TCP Vegas and will not be starved by TCP flows. When segment loss is detected, CTCP sending window is multiplicatively decreased ($win = win * (1 - \beta)$) which can avoid congestion collapse. Figure 1 shows a typical sending window vs. time graph of a CTCP flow which runs on a simulated bottleneck link (bandwidth: 100Mbps, delay: 50ms, packet error rate: 10^{-7}) with a DropTail queue whose size is 0.2 BDP. It is generated by NS 2.30 [10] patched with TCP-Linux [11]. This figure also indicates that CTCP has been correctly implemented in NS.

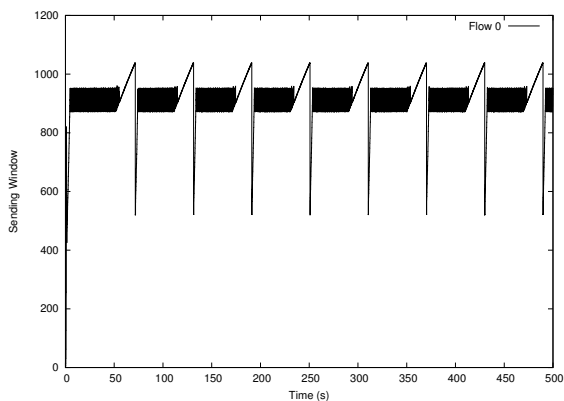


Fig. 1. A Typical CTCP Sending Window vs. Time Graph

In a word, with the assistance of queue delay, CTCP can well utilize network pipes with huge BDP while avoiding to starve TCP flows and avoiding to be starved by TCP flows. Since Microsoft has implemented CTCP in Windows Vista and CTCP can be enabled easily through a socket option, CTCP may be widely used in the Internet and it is necessary to evaluate its performance under many different network environments.

III. MOTIVATIONS AND EVALUATION METHODOLOGY

CTCP is a very new proposal and there is no much evaluation of this algorithm. Except performance evaluation carried out by the authors on their testbed [1], CTCP has also been evaluated on three Internet paths [12]. In [9][13], CTCP is evaluated and compared with their own proposals on testbeds.

Experiments on the Internet and testbed are very attractive for evaluating congestion control algorithms. These algorithms run on real computer systems. When carrying out experiments on the Internet, flows also pass the real network paths and interact with the real Internet cross traffic. Hence, the results are more credible. But it also has several constraints to evaluate HSCC algorithms on the Internet and testbed. First, although it is necessary to run HSCC algorithms on many Internet paths for thoroughly studying them, it is too dangerous since traffic generated during experiments may adversely affect users of the Internet. Second, for experiments on a testbed, it is very hard to accurately emulate very high speed networks due to hardware limitations. Hence, simulation is a necessary complement for studying HSCC algorithms thoroughly and systematically. With simulation, we can also focus on HSCC algorithms without considering the constraints of hardware and the interactions with other softwares.

In this paper, NS 2.30 [10] is adopted for studying CTCP. TCP-Linux [11] is patched so that we can study CTCP and compare it with other HSCC algorithms within a common TCP implementation. In this study, under many different network environments, CTCP is evaluated on the aspects of efficiency, fairness, and TCP friendliness. CTCP is also compared with other HSCC algorithms when it is necessary. Through this study, the parameters of CTCP and other HSCC algorithms use their default values which were carefully selected by their authors. For CTCP, that means $\alpha = 0.125$, $\beta = 0.5$, $\gamma = 30$, and $k = 0.75$. In order to mitigate the presence of phase effects, as was done in [4], for all flows, we set $overhead_$, NS parameter for node processing delay, to 8×10^{-6} . We also set $window_$, NS parameter for socket buffer, to 10000

packets so that the sending rate of these flows is only affected by congestion control.

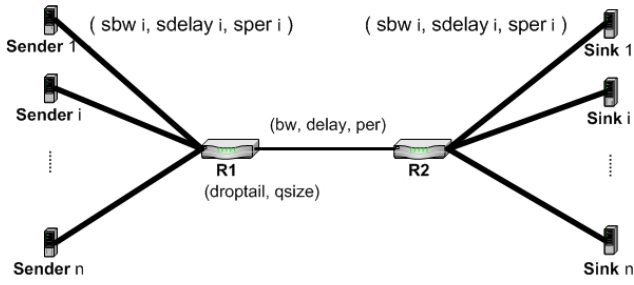


Fig. 2. Dumbbell Network Topology

Dumbbell network topology shown in figure 2 is widely used in this study. Side links, which connect end hosts and routers, are configured so that the link between $R1$ and $R2$ is the only bottleneck link. Specifically, their bandwidth (sbw_i) is always set to 1Gbps and packet error rate ($sper_i$) is always set to 0. As for propagation delay ($sdelay_i$), different values will be used when evaluating RTT fairness. The number of flows (n), congestion control algorithms used by these flows, and their start and end sequences also vary in different experiments. The details will be given when simulation results are presented.

As for the bottleneck link, it adopts DropTail queue and its bandwidth, propagation delay, queue size, and packet error rate ($bw, delay, qsize, per$) are set to different values. Table II summarizes the values adopted by these parameters. Hence, CTCP is studied on 180 different network configurations.

Parameter	Values
$bw : Mbps$	100, 200, 500, 1000
$delay : ms$	20, 50, 100
per	$10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$
$qsize : BDP$	0.1, 0.2, 0.5

TABLE II
NETWORK CONFIGURATIONS USED IN SIMULATION

IV. SIMULATION RESULTS AND ANALYSIS

In this section, simulation results are presented and analyzed in the order of metrics, efficiency, fairness, and TCP friendliness, etc.

A. Efficiency

Dumbbell topology in figure 2 is adopted and the propagation delay of side links, $sdelay_i$, is always set to 5ms. The utilization ratio of the bottleneck link is used to measure the efficiency of CTCP. Several groups of experiments have been carried out for this purpose.

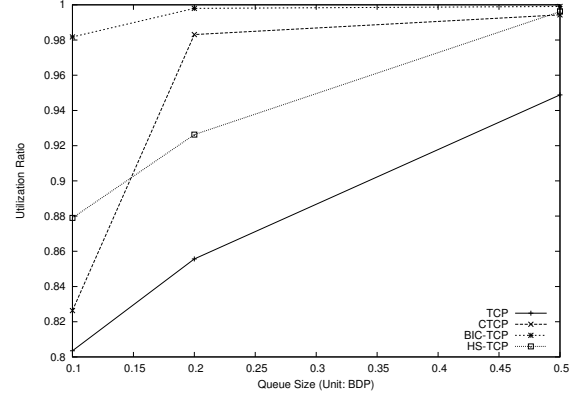


Fig. 6. Efficiency Comparison on a Specific Network Configuration ($bw=100Mbps, delay=20ms, per = 10^{-7}$)

1) *One Flow on Different Network Configurations:* In this group of experiments, only one CTCP flow runs for 500 seconds with the aim of investigating whether CTCP is scalable to bandwidth, delay, queue size, and packet error rate.

Figure 3 shows CTCP link utilization ratio on highly reliable network pipes, $per = 10^{-7}$, with different bandwidth, delay, and queue size. Figure 3 indicates that CTCP is very sensitive to queue size, $\frac{bw * 2 * delay * qsize}{Size_{data.pkt}}$. When queue is larger than γ , CTCP can detect congestion through queue delay and drive the network at the knee for a long time. If this condition is satisfied, CTCP is highly scalable and link utilization ratio is close to 1. When queue is smaller than γ , link utilization ratio can be quite slow. As shown in figure 6, in some cases, CTCP is much worse than HS-TCP and BIC-TCP. The reason is that when queue length is smaller than γ , CTCP flow only achieves binomial window growth, congestion occurs frequently, and multiple segments may be dropped in a congestion event since the increase step of win is large when congestion approaches. Hence, for each congestion event, sending window is reduced to half at least once and the throughput is quite low.

Since CTCP is very sensitive to queue length, its scalability to packet error rate is studied under two network configurations. Figure 4(a) shows the results when queue is huge and delay-based rules work well. It indicates that when packet error rate is very small, compared with HS-TCP and BIC-TCP, CTCP's link utilization ratio decreases faster with the increase of packet error rate. The reason is that CTCP's multiplicative decrease factor ($\beta = 0.5$) is much larger than HS-TCP and BIC-TCP. When packet error rate is moderate, CTCP's scalability to packet error rate is worse than BIC-TCP and is a little better than HS-TCP. Figure 4(b) shows the results when queue is smaller than γ . It indicates that CTCP is much worse than BIC-TCP, and CTCP is worse than HS-TCP when packet error rate is low. When packet error rate is high, CTCP can not detect congestion through queue delay

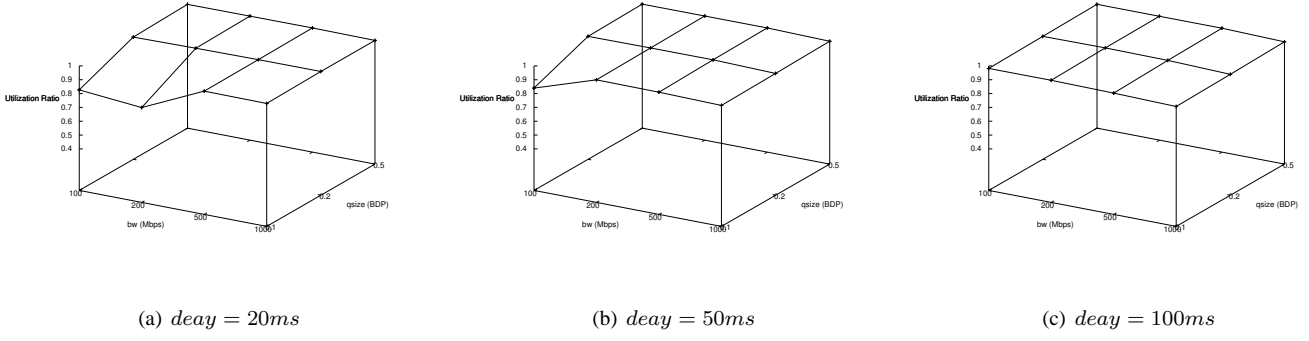


Fig. 3. CTCP Link Utilization Ratio on Highly Reliable Network Pipes ($per = 10^{-7}$) with Different Bandwidth, Delay, and Queue Size

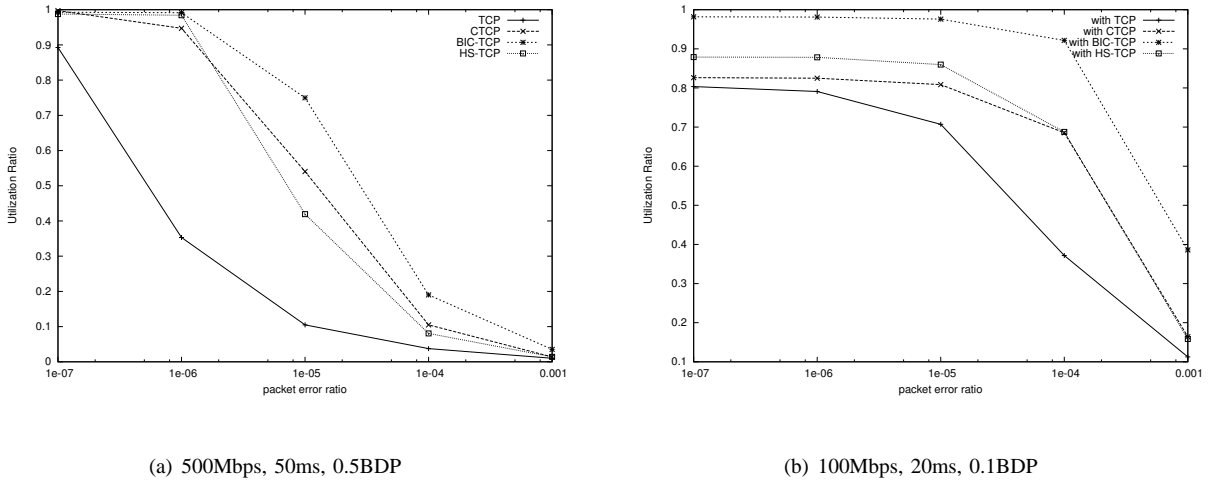


Fig. 4. Scalability to Packet Error Rate

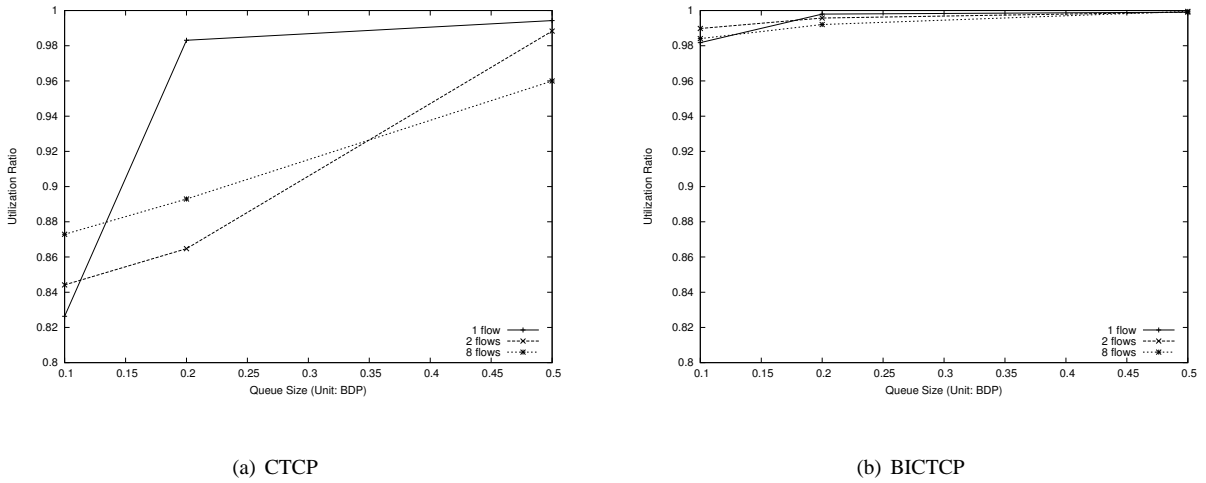


Fig. 5. Scalability to the Number of Flows ($bw=100Mbps, delay=20ms, per = 10^{-7}$)

and its sending window is binomially increased as if queue is small and congestion can not be detected through queue delay. Hence, in the following parts of the paper, we mainly evaluate CTCP on high reliable network pipes and make sure that it is evaluated under the two cases, $queue > \gamma$ and $queue < \gamma$.

2) *Multiple Flows on the Same Bottleneck Link*: According to above results, CTCP works very well when it can detect congestion through queue delay and works very bad when this condition is not satisfied. Since each CTCP flow tries to maintain γ packets in the bottleneck, the delay-based rules will be nullified when many CTCP flows pass through the same bottleneck link. Hence, CTCP is not scalable to the number of CTCP flows. Considering that Vista may be a popular operating system, many CTCP flows will exist and they will increase sending window binomially.

In order to evaluate this effect, two or eight flows will start simultaneously and run for 500 seconds on a specific bottleneck link ($bw=100Mbps$, $delay=20ms$, $per = 10^{-7}$) with different queue sizes. Figure 5(a) compares the results when one, two, or eight flows run on the same bottleneck link. It indicates when $qsize=0.1$, queue is too small and delay-based rules can not work for all cases. It also indicates that the larger the flow number is, the higher link utilization ratio is. The reason is that more flows can benefit more from statistical multiplexing. When $qsize=0.2$, link utilization ratio of eight flows is better than that of two flows due to the same reason. When $qsize=0.5$, queue is large enough for two flows to detect congestion through queue delay and they achieve high link utilization ratio. Although statistical multiplexing is helpful, eight flows achieve the lowest link utilization ratio since they still can not detect congestion through queue delay. Compared with BIC-TCP (shown in figure 5(b)), when CTCP flows can not detect congestion through queue delay, no matter the number of flows, their link utilization ratio is quite low.

In a word, CTCP's method for detecting congestion through queue delay is not scalable to the number of CTCP flows, and its binomial window growth function can not utilize network bandwidth well. It is necessary to solve this problem before CTCP is widely deployed.

B. Fairness

In this subsection, dumbbell topology in figure 2 is also used for evaluating fairness of CTCP. Three aspects, long-term fairness of CTCP flows with the same RTT, convergence issues, and RTT fairness are investigated.

1) *Long-Term Fairness*: In this group experiments, the propagation delay of all side links is always set to 5ms, and n (2 or 8) CTCP flows start at the same time and run on the bottleneck link for 500 seconds. Figure 7 shows Jain's Fairness Index of these CTCP flows's long-term throughput when they run on highly reliable network pipes with different configurations.

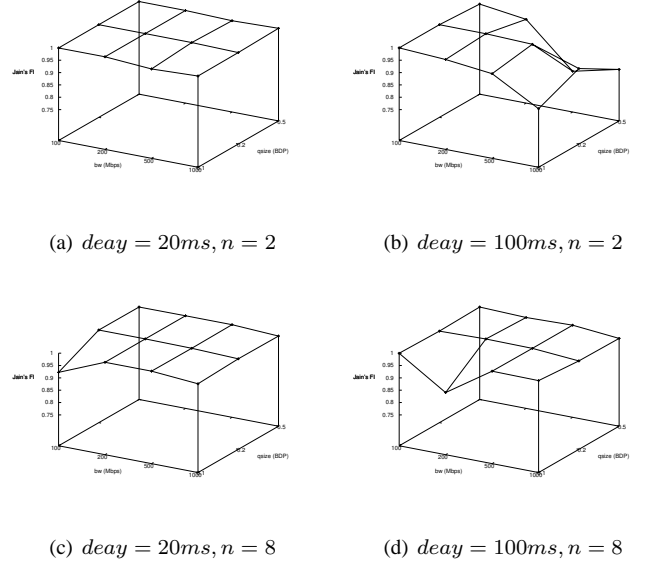


Fig. 7. CTCP Fairness Index on Highly Reliable Network Pipes ($per = 10^{-7}$) with Different Bandwidth, Delay, and Queue Size

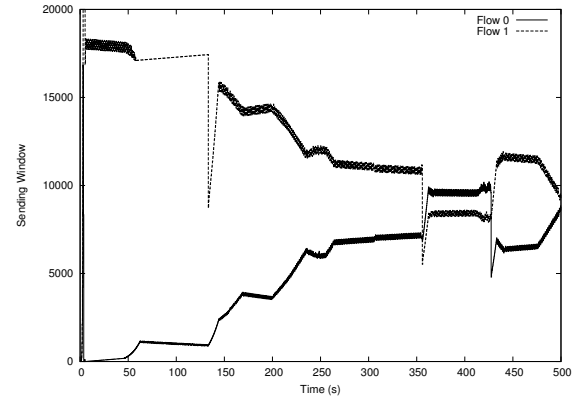


Fig. 8. Sending Window vs. Time Graph of two CTCP flows that competing a bottleneck link ($bw=100Mbps$, $delay=100ms$, $qsize=0.5BDP$, and $per = 10^{-7}$)

Figure 7(b) indicates that when BDP is huge, CTCP flows can not share the bandwidth fairly. CTCP's delay-based rules may be the root of this problem. When queue delay is detected by one CTCP flow, its win is reduced by $\zeta * \Delta$ (Δ is not much larger than γ). In the next RTT, its win will be increased by $\alpha * win$. If one flow's win is very large, $\alpha * win$ can be larger than $\zeta * \Delta$, the other flow has no opportunity to take bandwidth from this flow, and they can not converge at all. Another possible reason is that without pacing, CTCP senders can not sample queue delay evenly, and they may also judge network state at different times (the end of their each RTT). Hence, they may have different view of network state when

adjusting $cwnd$. By investigating further, we found that phase effect can occur in queue delay measurement. One flow can observe shorter queue delay and receives higher throughput for a long time. When packet loss ratio is very low, CTCP flows can stay in unfair state for quite a long time. Figure 8 illuminates this problem by plotting sending window vs. time graph of two CTCP flows which compete a highly reliable network pipe with huge BDP.

Figure 7(c) also indicate that when queue is small and there are quite a lot of flows, fairness index may be not very high. We find that the binomial window growth function does not assure absolute fairness when segment loss is highly synchronized. In figure 7(d), we find that when eight flows compete a specific bottleneck link ($bw=200\text{Mbps}$, $delay=100\text{ms}$, $per = 10^{-7}$, and $qsize=0.1\text{BDP}$), their fairness index is exceptionally low. By plotting their throughput vs. time graph (as shown in figure 9), we find that for some unknown reasons, CTCP flows may leave the converged state and some flows are starved.

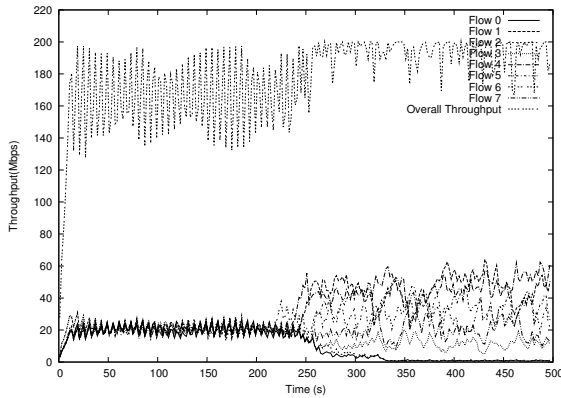


Fig. 9. Throughput vs. Time Graph of eight CTCP flows that competing a bottleneck link ($bw=200\text{Mbps}$, $delay=100\text{ms}$, $qsize=0.1\text{BDP}$, and $per = 10^{-7}$)

2) *Convergence Issues*: In this group of experiments, the propagation delay of all side links is also set to 5ms and two CTCP flows will run on the same bottleneck link. But the first flow starts at 0 second and ends at 700th second. The second flow starts at 100th second and ends at 600th second. Although convergence speed can be used as the metric, we find that it is not enough to characterize the complex convergence behaviors of CTCP. Hence, the throughput vs. time graph of several typical network pipes are plotted in figure 10.

Figure 10 indicates that when the second flow leaves at 600th second, the first flow can fully utilize the bottleneck link in a very short time. When the second flow starts to transmit at 100th second, the two CTCP flows show very complex convergence behaviors. Figure 10(a) indicates that CTCP flows can converge well when delay-based rules can not work due to small buffer size. But the cost is the low link utilization ratio. Figure 10(b) shows the interaction between the

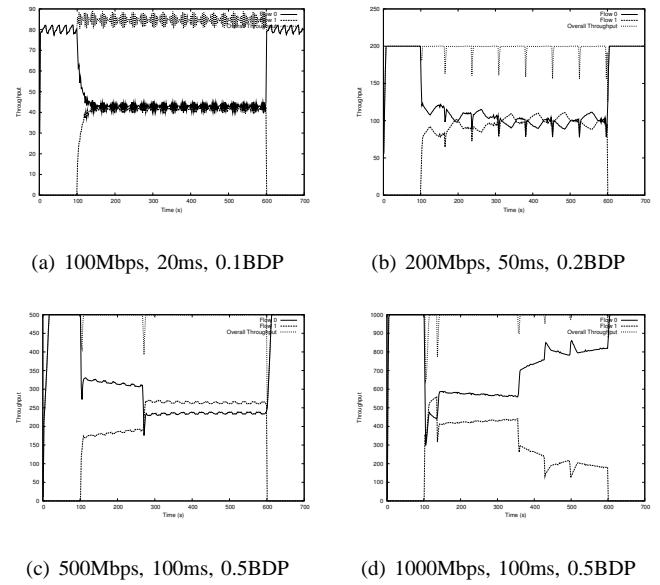


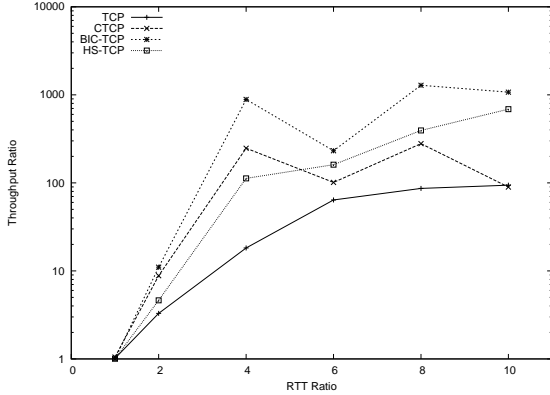
Fig. 10. CTCP Convergence Issues on Highly Reliable Network Pipes ($per = 10^{-7}$) with Different Bandwidth, Delay, and Queue Size

rules used for updating $cwnd$ and $dwnd$. At 120th seconds, although flow 1 consumes more bandwidth than flow 2, it does not continue give bandwidth to flow 2 because its $dwnd$ has been reduced to zero. This interaction is affected by the BDP of network pipe and the start time of the second flow. Hence, the convergence behavior is very complex and unpredictable. Figure 10(c) indicates that when BDP is huge, delay-based rules give little help for the convergence of CTCP flows. Figure 10(d) indicates that when BDP is huge and delay-based rules give little help to convergence, asynchronous packet loss can bring disastrous results and worsen the unfairness.

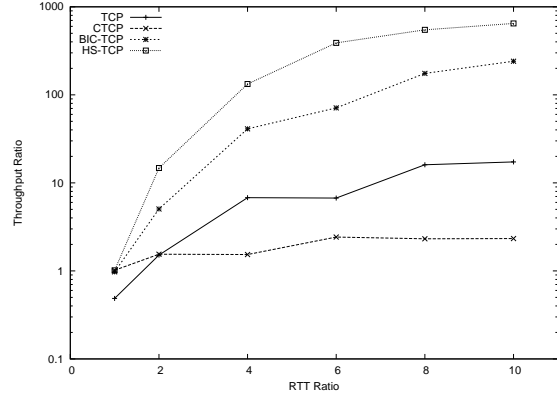
3) *RTT Fairness*: In order to evaluate RTT fairness, two CTCP flows start at the same time and run on the bottleneck link for 500 seconds. The propagation delay of bottleneck link is set to 20ms. The propagation delay of side links used by the first flow is always set to 5ms. As for the second flow, the propagation delay of side links is set to 5, 20, 50, 80, 110, and 140ms so that the RTT ratio is 1, 2, 4, 6, 8, 10. $\frac{Th_{1st\ flow}}{Th_{2nd\ flow}}$ is used as the metric.

Figure 11 compared CTCP with TCP, BIC-TCP, and HS-TCP in the aspect of RTT fairness. Figure 11(b) indicates that when queue is large enough, CTCP is very good on RTT fairness. But figure 11(b) indicates that when queue is small and CTCP flows can not detect congestion through queue delay, CTCP shows much worse RTT fairness.

In summary, due to the complex interactions between $cwnd$ and $dwnd$, the convergence issues of CTCP are very complex and unpredictable. When segment loss is asynchronous, the situation can be even worse. When BDP is huge and link is highly reliable, the delay-based rules of CTCP gives little help to the convergence of CTCP flows,



(a) 100Mbps, 20ms, 0.1BDP



(b) 1000Mbps, 20ms, 0.5BDP

Fig. 11. RTT Fairness on Highly Reliable Network Pipes ($per = 10^{-7}$)

CTCP flows can stay in unfair state for a long time, and CTCP only achieve theoretical fairness since many flows die before acquiring fair share of bandwidth. When queue is small, RTT fairness of CTCP also becomes very bad.

C. TCP Friendliness

In order to evaluate TCP friendliness, the propagation delay of side links is always set to 5ms. One CTCP flow and one TCP flow start at the same time and run for 500 seconds on the bottleneck link. The average throughput of two TCP flows, that run on the bottleneck link with same configuration, is used as the baseline (Thr_{base}). And the throughput ratio, $\frac{Thr_{TCP\ flow\ that\ competing\ with\ CTCP\ flow}}{Thr_{base}}$, is used as the metric. Here, the metric can be larger than 1.

Figure 12 shows the results under different network configurations. It indicates that when queue is large enough, CTCP is friendly to TCP flow. The TCP flow even acquires more bandwidth than the bandwidth acquired when it competes with TCP flow. But when queue is small and CTCP flow can not detect congestion through queue delay, it still can starve TCP flow. Figure 12(c) shows that CTCP TCP friendliness is exceptionally low on a specific network pipe ($bw=1000Mbps$, $delay=100ms$, $qsize=0.1BDP$, $per = 10^{-7}$). By investigating further, we find the reason is that one segment of TCP flow is corrupted very early and TCP flow increase its sending rate very slow.

We had also investigated several other scenarios (TCP flow starts first, CTCP Flow starts first, three CTCP flows and one TCP flow, three TCP flow and one CTCP flow) and their results are similar to Figure 12. We had also investigated TCP friendliness in detail under a simulated Internet scenario. Due to space limitation, the results are not presented here.

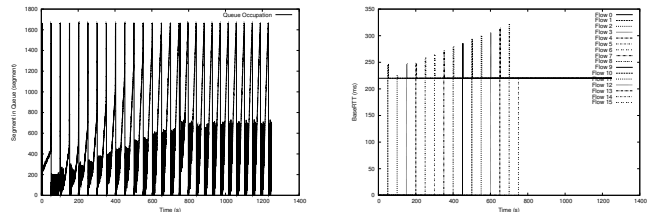
In a word, CTCP is friendly to TCP flow. But when

congestion can not be detected through queue delay, TCP flows still can be starved.

D. Other Issues

In this section, we investigated several problems found during study TCP Vegas, such as persistent congestion and re-routing state [6].

When Vegas flows join a network one after another. Queue delay will increase and the network enters into persistent congested state. $baseRTT$ observed by new flows is normally larger than that of old flows, and new flows will get more bandwidth than old flows. We investigate this problem by simulating 16 CTCP flows join a specific network pipe ($bw = 200Mbps$, $delay = 100ms$, $qsize = 0.5BDP$, $per = 10^{-7}$) one after another. Figure 13 indicates that average queue occupation does increase, but CTCP flows can observe the same $baseRTT$ in a short time and there is no unfairness to old flows.



(a) Queue Occupation

(b) $baseRTT$ Fig. 13. CTCP Persistent Congestion on a Specific Network Pipe ($bw = 200Mbps$, $delay = 100ms$, $qsize = 0.5BDP$, $per = 10^{-7}$)

When re-routing occurs and the RTT is increased, Vegas flow keep decreasing its sending rate. For CTCP flows, as shown in figure 14, they switch to regular TCP. Hence, CTCP

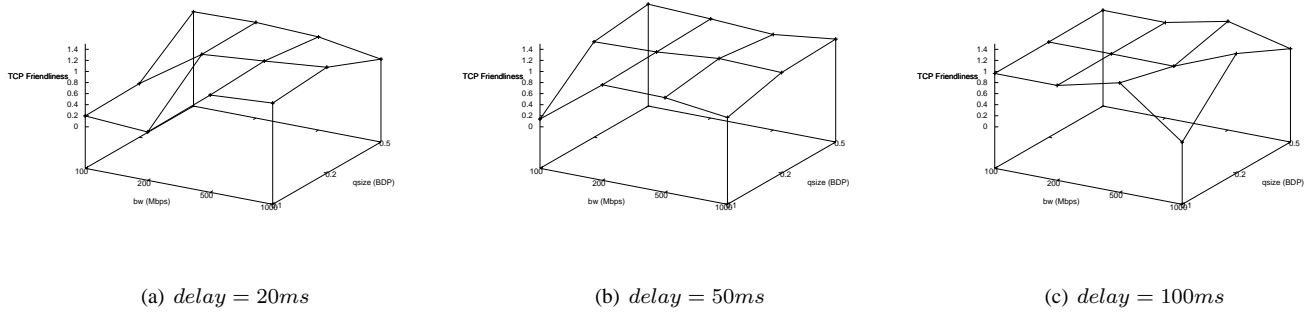


Fig. 12. TCP Friendliness of CTCP under High Reliable Network Pipes

is better than Vegas. But when BDP is huge, bandwidth can not be well utilized and some improvements are needed.

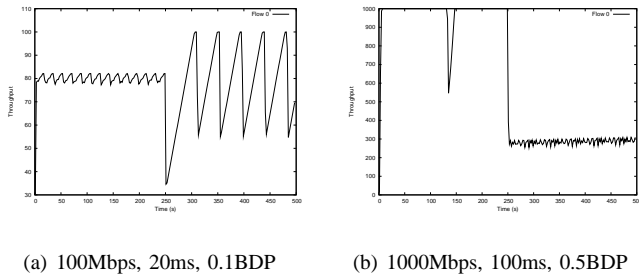


Fig. 14. One CTCP Flow on Highly Reliable Network Pipes ($per = 10^{-7}$) Whose Delay is Doubled at 250th Second

CTCP is also liable to reverse traffic. Queue delay on the backward path is still regarded as a congestion signal of the forward path. As reported in [14], CTCP flows can be starved by loss based HSCC algorithms, such as BIC-TCP.

In a word, these issues should also be solved or well argued before deploying CTCP.

V. CONCLUSION

In this paper, we systematically studied the recently proposed Compound TCP by simulation, and a number of issues are of practical concern highlight. Among these, the issue, that CTCP queue delay detection method is not scalable to flow number and binomial window growth itself performs very bad, seems most worrying. Its unfairness on network pipes with huge BDP also brings severe challenge to the deployment of Compound TCP.

REFERENCES

- [1] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *INFOCOM*, 2006.
- [2] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM SIGCOMM CCR*, vol. 19, pp. 56–71, Oct. 1989.
- [3] S. Floyd, "Highspeed tcp for large congestion windows," RFC 3649, Dec. 2003.
- [4] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long distance networks," in *INFOCOM*, 2004.
- [5] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994.
- [6] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," in *IEEE INFOCOM*, 1999.
- [7] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM*, 1988.
- [8] C. Jin, D. X. Wei, and S. H. Low, "Fast tcp: motivation, architecture, algorithms, performance," in *INFOCOM*, 2004.
- [9] K. Kaneko, T. Fujikama, Z. Su, and J. Katto, "Tcp-fusion: A hybrid congestion control algorithm for high-speed networks," in *PFLDnet Workshop*, 2007.
- [10] "Ns2 network simulator," <http://www.isi.edu/nsnam/ns/>.
- [11] 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.
- [12] Y.-T. Li, "Evaluation of tcp congestion control algorithms on the windows vista platform," Tech. Report SLAC-TN-06-005, Stanford University, Tech. Rep., June 2006.
- [13] A. Baiocchi, A. P. Castellani, and F. Vacirca, "Yeah-tcp: Yet another highspeed tcp," in *PFLDnet Workshop*, 2007.
- [14] K. Munir, M. Welzl, and D. Damjanovic, "Linux beats windows! c or the worrying evolution of tcp in common operating systems," in *PFLDnet Workshop*, 2007.