Network Working Group
Internet Draft
Intended status: Experimental
November 3, 2008
Expires: April 2009

M. Sridharan
Microsoft
K. Tan
Microsoft Research
D. Bansal
D. Thaler
Microsoft

Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks

draft-sridharan-tcpm-ctcp-02.txt

Status of this Memo

Copyright Notice

Abstract

Compound TCP (CTCP) is a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows. This document describes the Compound TCP algorithm in detail, and solicits experimentation and feedback from the wider community. The key idea behind CTCP is to add a scalable delay-based component to the

standard TCP's loss-based congestion control. The sending rate of CTCP

is controlled by both loss and delay components. The delay-based component has a scalable window increasing rule that not only efficiently uses the link capacity, but on sensing queue build up, proactively reduces the sending rate.

Table of Contents

1. Introduction

In this document, we collectively refer to any TCP congestion control algorithm that employs a linear increase function for congestion control, including TCP Reno and all its variants as Standard TCP. This document describes Compound TCP, a modification to TCP's congestion control mechanism for fast, long-distance networks. The standard TCP congestion avoidance algorithm employs an additive increase and multiplicative decrease (AIMD) scheme, which employs a conservative linear growth function for increasing the congestion window and multiplicative decrease function on encountering a loss. For a high-speed and long delay network, it takes standard TCP an unreasonably long time to recover the sending rate after a single loss event [RFC2581, RFC3649]. Moreover, it is well-known now that in a steady-state environment, with a packet loss rate of p, the current standard TCP's average congestion window is inversely proportional to the square root of the packet loss rate [RFC2581,PADHYE]. Therefore, it requires an extremely small packet loss rate to sustain a large window. As an example, Floyd et al. [RFC3649], pointed out that on a 10Gbps link with 100ms delay, it will roughly take one hour for a standard TCP flow to fully utilize the link capacity, if no packet is lost or corrupted. This one hour error-free transmission requires a packet loss rate of around 10^-11 with 1500-byte size packets (one packet loss over 2,600,000,000 packet transmission!), which is not practical in today's networks.

There are several proposals to address this fundamental limitation of TCP. One straightforward way to overcome this limitation is to modify TCP's increase/decrease rule in its congestion avoidance stage. More specifically, in the absence of packet loss, the sender increases congestion window more quickly and decreases it more gently upon a packet loss. In a mixed network environment, the aggressive behavior of such an approach 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 a result, the aggressive variants will cause much more self-induced packet losses on bottleneck links, and push back the throughput of the regular TCP flows.

Then there is the class of high-speed protocols which use variances in RTT as a congestion indicator (e.g., [AFRICA,FAST]). Such delay-based approaches are more-or-less derived from the seminal work of TCP-Vegas [VEGAS]. An increase in RTT is considered an early indicator of congestion, and the sending rate is reduced to avoid buffer overflow. The problem in this approach comes when delay-based and loss-based flows share the same bottleneck link. While the delay-based flows respond to increases in RTT by cutting its sending rate, the loss-based flows continue to increase their sending rate. As a result a delay-based flow obtains far less bandwidth than its fair share. This weakness is hard to remedy for purely delay-based approaches.

The design of Compound TCP is to satisfy the efficiency requirement and the 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 will not only adversely affect standard TCP flows but will also cause instability. As noted above, delay-based approaches already have the nice property of adjusting aggressiveness based on the link utilization, which is observed by the end-systems as an increase in RTT. CTCP incorporates a scalable delay-based component into the standard TCP's congestion avoidance algorithm. Using the delay component as an automatic tuning knob, CTCP is

scalable
yet TCP friendly.

2. Design Goals

The design of CTCP is motivated by the following requirements:

- Improve throughput by efficiently using the spare capacity in the network
- Good intra-protocol fairness when competing with flows that have different RTTs
- Should not impact the performance of standard TCP flows sharing the same bottleneck
- No additional feedback or support required from the network

CTCP can efficiently use the network's resources and achieve high link utilization. The aggressiveness can be controlled by adopting a rapid increase rule in the delay-based component. We choose CTCP to have similar aggressiveness as HighSpeed TCP [RFC3649]. Our design choice is motivated by the fact that HSTCP has been tested to be aggressive enough in real world networks while at the same time, not exhibiting any severe issues in deployment or testing experiences. and is now an experimental IETF RFC. We also wanted an upper bound on the amount of unfairness to standard TCP flows. However, as shown later, CTCP is able to maintain TCP friendliness under high statistical multiplexing and also while traversing poorly buffered links. CTCP has similar or, in some cases, improved RTT fairness compared to standard TCP. As we will demonstrate later this is due to the fact that the amount of backlogged packets for a connection is independent of the RTT of the connection. Even though CTCP does not require any feedback from the network, CTCP works well in ECN capable environments. There is also no expectation on the queuing algorithm deployed in the routers.

As is the case with most high-speed variants today, CTCP does not modify the slow-start behavior of standard TCP. We agree to the belief that ramping-up faster than slow-start without additional information from the network can be harmful. During slow start, CTCP uses standard TCP congestion window (cwnd) and does not use any additional delay component. Just like standard TCP, it exits slow start when either a loss happens or congestion window (cwnd) reaches ssthresh.

Similar to HSTCP, to ensure TCP compatibility, CTCP's scalable component uses the same response function as Standard TCP when the current congestion window is at most Low_Window. CTCP sets Low_Window to 38 MSS-sized segments, corresponding to a packet drop rate of 10^-3 for TCP.

3. Compound TCP Control Law

CTCP modifies Standard TCP's loss-based control law with a scalable delay-based component. To do so, a new state variable is introduced in current TCP Control Block (TCB), namely dwnd (Delay Window), which controls the delay-based component in CTCP. The conventional congestion window, cwnd, remains untouched, which controls the loss-based component in CTCP. Thus, the CTCP sending window now is controlled by both cwnd and dwnd. Specifically, the TCP sending window (wnd) is now calculated as follows:

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

where awnd is the advertised window from the receiver.

cwnd is updated in the same way as regular TCP in the congestion avoidance phase, i.e., cwnd is increased by 1 MSS every RTT and halved when a packet loss is encountered. The update to dwnd will be explained in detail later in this section. The combined window for CTCP from (1) above allows up to (cwnd + dwnd) packets in one RTT to be injected into the network. Therefore, the increment of cwnd on the arrival of an ACK is modified accordingly:

```
  cwnd = cwnd + 1/(cwnd+dwnd)               (2)
```

Some implementations may choose to use FlightSize (as defined in RFC 2581) to handle the receiver limited or the application limited case. As stated above, CTCP retains the same behavior during slow start. When a connection starts up, dwnd is initialized to zero while the connection is in slow start phase. Thus the delay component is only activated when the connection enters congestion avoidance. The delay-based algorithm has the following properties. It uses a scalable increase rule when it infers that the network is under-utilized. It also reduces the sending rate when it senses incipient congestion. By reducing its sending rate, the delay-based component yields to competing TCP flows and ensures TCP fairness. It reacts to packet losses, again by reducing its sending rate, which is necessary to avoid congestion collapse. CTCP's control law for the delay-based component is derived from TCP Vegas. A state variable, called basertt tracks the minimum round trip delay seen by a packet over the network path. The CTCP sender also maintains a smoothed RTT srtt, updated as specified in [RFC2988]. Basertt is not used till the delay component is activated so basertt can be initialized to the smoothed rtt value that the sender already computed. Basertt MUST be uninitialized and MUST be re-measured if a retransmission timeout occurs, as the network conditions may have changed. We provide some guidance on RTT sampling in Section 6 as robust RTT sampling is key to how CTCP implementations perform.

The number of backlogged packets of the connection is estimated using,

```
  expected (throughput) = wnd/basertt
  actual (throughput) = wnd/srtt
  diff = (expected - actual) * basertt
```

The expected throughput gives the estimation of throughput CTCP gets if it does not overrun (induce queueing on) the network path. The actual throughput stands for the throughput CTCP sender really gets. Using this, the amount of data backlogged in the bottleneck queue (diff) can be calculated. Congestion is detected by comparing diff to a threshold gamma. If diff < gamma, the network path is assumed to be underutilized; otherwise the network path is assumed to be congested and CTCP should gracefully reduce its window.

It is to be noted that a connection should have at least gamma packets backlogged in the bottleneck queue to be able to detect incipient congestion. This motivates the need for gamma to be small since the implication is that even when the bottleneck buffer size is small, CTCP will react early enough to ensure TCP fairness. On the other hand, if gamma is too small compared to the queue size, CTCP will falsely detect congestion and will adversely affect the throughput. Choosing the appropriate value for gamma could be a problem because this parameter depends on both network configuration and the number of concurrent flows, which are generally unknown to the end-systems. Section 5 presents an effective way to automatically estimate gamma.

The increase law of the delay-based component should make CTCP more scalable in high-speed and long delay pipes. We choose a binomial function to increase the delay window [BAINF01]. As explained in the next section we have modeled the response function for CTCP to have comparable scalability to HighSpeed TCP. Since there is already a loss-based component in CTCP, the delay-based component needs to be designed to only fill the gap. The control law for CTCP's delay component can be summarized as follows:

```
 dwnd(t+1) =
     dwnd(t) + alpha*dwnd(t)^k - 1,     if diff < gamma  (3)
     dwnd(t) - eta*diff,                if diff >= gamma (4)
     dwnd(t)(1-beta),          on packet loss   (5)
```

where alpha = 1/8, beta = 1/2, eta = 1 and k = 0.75. Note that dwnd MUST be measured in packets to match the response function in Section 4. Equation (3) shows that in the increase phase, dwnd only needs to increase by (alpha*dwnd(t)^k - 1) packets, since the loss-based component cwnd will also increase by 1 packet. When a packet loss occurs (detected by three duplicate ACKs), dwnd is set to the difference between the desired reduced window size and that can be provided by cwnd. The rule in equation (4) is very important to preserve good RTT and TCP fairness. Eta defines how rapidly the delay component should reduce its window when congestion is detected. Note that dwnd MUST never be negative, so the CTCP window is lower bounded by its loss-based component, which is same as Standard TCP.

If a retransmission timeout occurs, dwnd should be reset to zero and the delay-based component is disabled. This is because after a timeout, the TCP sender enters slow-start phase. After the CTCP sender exits the slow-start recovery state and enters congestion avoidance, dwnd control is activated again.

4. Compound TCP Response Function

The TCP response function provides a relationship between TCP's average congestion window w in MSS-sized segments as a function of the steady-state packet drop rate p. To specify a modified response function for CTCP, we use the analytical model in [CTCPI06] to derive a relationship between w and p. Based on this model, the response function for CTCP provides the following relationship between w and p,

$$w \sim .1/(p^{(1/(2-k))}) \qquad (6)$$

As explained earlier we modeled the response function for CTCP to have comparable scalability to HighSpeed TCP. The response function for HighSpeed TCP is

$$w \sim .1/p^{0.835} \qquad (7)$$

Comparing (6) and (7) we get k to be around 0.8. Since it's difficult to implement an arbitrary power we choose k = 0.75 which can be implemented using a fast integer algorithm for square root. Based on extensive experimentation, we chose alpha = 1/8, beta = 1/2, and eta = 1. **Substituting the above values for alpha, beta and k in (6) we get** the following response function for CTCP,

$$w = 0.255/p^{0.8} \qquad (8)$$

The response function for CTCP is compared with HSTCP and is illustrated in Table 1 below.

	CTCP	HSTCP
Packet Drop Rate P	Congestion Window W	Congestion Window W
10^-3	64	38
10^-4	404	263
10^-5	2552	1795
10^-6	16107	12279
10^-7	101630	83981
10^-8	641245	574356
10^-9	4045987	3928088
10^-10	25528453	26864653

Table 1: TCP Response function for CTCP & HSTCP

The values in Table 1 illustrate that our choice of parameters makes CTCP slightly more aggressive than HSTCP in moderate and low packet loss rates but approaches HSTCP for larger windows. The reason we choose to do this is because unlike HighSpeed TCP, CTCP's delay control is capable of scaling back on detecting incipient congestion. As a result, we expect CTCP to be more TCP friendly than HighSpeed TCP. We show that this is in fact the case even under low buffering conditions in the presence of high statistical multiplexing. The fairness considerations and choice of gamma are detailed in Sections 5 and 6.

5. Automatic Selection of Gamma

To effectively detect early congestions, CTCP requires estimating the backlogged packets at the bottleneck queue and compares this estimate to a pre-defined threshold gamma. However, setting this threshold gamma is particularly difficult for CTCP (and for many other similar delay-based approaches) because gamma largely depends on the network configuration and the number of concurrent flows that compete for the same bottleneck link. Such flows are, unfortunately, unknown to end-systems. Based on experimentation over varying conditions we originally selected gamma to be 30 packets. This value appeared to provide a good tradeoff between TCP fairness and throughput. However a fixed gamma can still result in poor TCP friendliness over under-buffered network links. One naive solution is to choose a very small value for gamma. However this can falsely detect congestion and adversely affect throughput. To address this problem, we instead use a method called tuning-by-emulation to dynamically adjust gamma. The basic idea is to estimate the backlogged packets of a Standard TCP flow along the same path by simultaneously emulating the behavior of a Standard TCP flow. Based on this, gamma is set so as to ensure good TCP-friendliness. CTCP can then automatically adapt to different network configurations (i.e., buffer provisioning) and also concurrent competing flows.

To ensure the effectiveness of incipient congestion detection, our analytical model on CTCP shows that gamma should at least be less than B/(m+l), where B is the bottleneck buffer and m and l represent the number of concurrent Standard TCP flows and CTCP flows, respectively, that are competing for the same bottleneck link [CTCPI06][CTCPP06] [CTCPT]. Generally, both B and (m+l) are unknown to end-systems. It is very difficult to estimate these values from end-systems in real-time, especially the number of flows, which can vary significantly over time. Fortunately there is a way to directly estimate the ratio B/(m+l), even though the individual variables B and (m+l) are hard to estimate. Let's first assume there are (m+l) regular TCP flows in the network. These (m+l) flows should be able to fairly share the bottleneck capacity in steady state. Therefore, they should also get roughly equal shares of the buffers at the bottleneck, which should equal to B/(m+l). For such

a Standard TCP flow, although it does not know either B or (m+l), it can still infer B/(m+l) easily by estimating its backlogged packets, which is a rather mature technique widely used in many delay-based protocols. This brings us to the core idea of CTCP's algorithm; CTCP lets the sender emulate the congestion window of a Standard TCP flow. Using this emulated window, we can estimate the buffer occupancy (diff_reno) for a Standard TCP flow. Diff_reno can be regarded as a conservative estimate of B/(m+l) assuming that the high speed flow is more aggressive than Standard TCP. By choosing gamma <= diff_reno, we can ensure TCP fairness.

The implementation is actually fairly trivial. This is because CTCP already emulates Standard TCP as the loss-based component. We can simply estimate the buffer occupancy of a competing Standard TCP flow from state that CTCP already maintains. We choose an initial gamma = 30

and diff_reno is calculated as follows,

```
 expected_reno (throughput) = cwnd/basertt
 actual_reno (throughput) = cwnd/srtt
 diff_reno = (expected - actual) * basertt
```

The difference between diff_reno and diff is simply that diff_reno is computed only using the loss-based component cwnd. Since Standard TCP reaches its maximum buffer occupancy just before a loss, CTCP uses the

diff_reno value computed in the previous round to calculate the gamma for the next round. A round corresponds to the time it takes for one window of data
to be acknowledged. It typically corresponds to one RTT. Whenever a loss

happens, gamma is chosen to be less
than diff_reno and the sample values of gamma are updated using a standard exponentially weighted moving average. The pseudocode to calculate gamma is shown below. Here a round tracks every window worth of data. Section 7 provides more details on how to maintain a round.

```
  Initialization:
    diff_reno = invalid;
     Gamma = 30;

  End-of-Round:

     expected_reno = cwnd / baseRTT;
     actual_reno = cwnd / RTT;
     diff_reno = (Expected_reno-Actual_reno)*baseRTT;

  On-Packet-Loss:

  If diff_reno is valid then
     g_sample = 3/4*Diff_reno;
     gamma = gamma*(1-lamda)+ lamda*g_sample;
     if (gamma < gamma_low)
       gamma=gamma_low;
     else if (gamma > gamma_high)
       gamma=gamma_high;
     fi
     diff_reno = invalid;
  fi
```

The recommended values for gamma_low and gamma_high are 5 and 30 respectively. diff_reno is set to invalid to prevent using stale diff_reno data when there are consecutive losses between which no samples were taken.

6. Implementation Issues

CTCP has been implemented on Microsoft Windows and there has been extensive testing on production links and in Windows Beta deployments.

The first challenge is to design a mechanism that can precisely track the changes in round trip time with minimal overhead, and can scale well to support many concurrent TCP connections. Naively taking RTT samples for every packet will obviously be an over-kill for both CPU and system memory, especially for high-speed and long distance networks where the congestion window can be very large. Therefore, CTCP needs to limit the number of samples taken, but without compromising on accuracy. In our implementation, we only take up to M samples per window of data. M is chosen to scale with the round trip delay and window size.

In order to further improve the efficiency in memory usage, we have developed a memory allocation mechanism to dynamically allocate sample buffers from a kernel fixed-size per-processor pool. The size should be chosen as a function of the available system memory. As the window size increases, M can be updated so that the samples are uniformly distributed over the window. As M gets updated, more memory blocks are allocated and linked to the existing sample buffers. If the sending rate changes, either due to network conditions or due to application behavior, the sample blocks are reclaimed to the global memory pool. This dynamic buffer management ensures the scalability of our implementation, so that it can work well even in a busy server which could host tens of thousands of TCP connections simultaneously. Note that it may also require a high-resolution timer to time RTT samples.

The rest of the implementation is rather straightforward. We add two new state variables into the standard TCP Control Block, namely dwnd and basertt (described in Section 3). Following the common practice of high-speed protocols, CTCP reverts to standard TCP behavior when the window is small. Delay-based component only kicks in when cwnd is larger than some threshold, currently set to 38 packets assuming 1500 byte MTU. dwnd is updated at the end of each round. Note that no RTT sampling and dwnd update happens during the loss recovery phase. This is because the retransmission during the loss recovery phase may result

in inaccurate RTT samples and can adversely affect the delay-based control.

7. Deployment Issues

There are several variations of TCP proposed for high speed and long delay networks. We do not claim Compound TCP to be the best nor the most optimal algorithm. However, based on extensive testing via simulations and experimentation including those on production links as well as beta deployments of a reasonable scale, we believe that Compound TCP satisfies the design considerations outlined earlier in this document. It effectively uses spare bandwidth in high speed networks, achieves good intra-protocol fairness even in the presence of differing RTTs and does not adversely impact standard TCP. Furthermore, Compound TCP does not require any changes or any new feedback from the network and is deployable over the current Internet in an incremental fashion. It interoperates with Standard TCP and requires support only on the send side of a TCP connection for it to be used.

We also note that similar to High Speed TCP, in environments typical of much of the current Internet, Compound TCP behaves exactly like Standard TCP. This it does by ensuring that it follows the standard TCP algorithm without any modification any time the congestion window is less than 38 packets. Only when the congestion window is greater than

38 packets does the delay-based component of Compound TCP get invoked.

Thus, for example for a connection with an RTT of 100ms, the end-to-end bandwidth must be greater than 4.8Mbps for CTCP to have any difference in its response to network conditions compared to standard TCP.

Further, we do not believe that the deployment of Compound TCP would block the possible deployment of alternate experimental congestion control algorithms such as Fast TCP [FAST] or CUBIC [CUBIC]. In particular, Compound TCP's response has a fallback to a loss-based function that has characteristics very similar to HS-TCP or N parallel TCP connections.

8. Security Considerations

CTCP modifies the congestion control algorithm of TCP protocol by adding a delay based component while keeping all other aspects of the protocol intact. Hence, any additional security considerations for CTCP are limited to the security considerations for the delay based aspect of the CTCP algorithm.

There are a few possible security considerations for the delay based component of CTCP. A receiver can explicitly delay the acknowledgements or it can proactively acknowledge packets. In the former case dwnd increase would be slower and the throughput would be no worse than standard TCP. In the latter case the sender may end up sending traffic at a higher rate. However as the packets are proactively acknowledged the sender will update its basertt to be much lower than the actual RTT. So any increases in measured RTT will be perceived as congestion. Further, sender can implement additional mitigations to detect such a malicious receiver eg by detecting if spurious acknowledgements are being acknowledged too soon i.e. faster than RTT and without actually receiving the packet. The delay measurements for CTCP are derived at the sender-side only, without relying on timestamps. This mitigates possible attacks where receiver manipulates the timestamps echoed back to the sender.

9. IANA Considerations

There are no IANA considerations regarding this proposal.

10. Conclusions

This document proposes a congestion control algorithm for TCP for high speed and long delay networks. By introducing a delay-based component in addition to a standard TCP-based loss component, Compound TCP is able to detect and effectively use spare bandwidth that may be available on a high speed and long delay network. Furthermore, the delay-based component detects the onset of congestion early and gracefully reduces the sending rate. The loss-based component, on the other hand, ensures there is an effective response to losses in network while in the absence of losses, keeps the throughput of CTCP lower bounded by TCP Reno. Thus, CTCP is not timid, nor does it induce more self-induced packet loss than a single standard TCP flow. Thus Compound TCP is efficient in consuming available bandwidth while being friendly to standard TCP. Further, the delay component does not have any RTT bias thereby reducing the RTT bias of the Compound TCP vis-a-vis standard TCP.

Compound TCP has been implemented as an optional component in Microsoft Windows Vista. It has been tested and experimented through broad Windows Vista beta deployments where it has been verified to meet its objectives without causing any adverse impact. The Stanford Linear Accelerator Center (SLAC) has also evaluated Compound TCP on production links. Based on testing and evaluation done so far, we believe Compound TCP is safe to deploy on the current Internet. We welcome additional analysis, testing and evaluation of Compound TCP by Internet community at large and continue to do additional testing ourselves.

11. Acknowledgments

The authors would like to thank Jingmin Song for all his efforts in evaluating the algorithm on the test beds. We are thankful to Yee-ting Lee and Les Cottrell for testing and evaluation of Compound TCP on Internet2 links [SLAC]. We would like to thank Sanjay Kaniyar for his insightful comments and for driving this project in Microsoft. We are also thankful to the Microsft.com data center staff who helped us evaluate Compound TCP on their production links. In addition, several folks from the Internet research community who attended the High-Speed TCP Summit at Microsoft [MSWRK] have provided valuable feedback on Compound TCP. We would like to thank CTCP reviewers at ICCRG for their valuable feedback; specifically we would like to thank Lachlan Andrew and Doug Leith for their thorough review and excellent feedback. Finally, we are thankful to the Windows Vista program beta participants who helped us test and evaluate CTCP.

12. References

12.1. Normative References

[CTCPI06] K. Tan, Jingmin Song, Qian Zhang, Murari Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks", in IEEE Infocom, April 2006, Barcelona, Spain.

[RFC2581] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

12.2. Informative References

[AFRICA]   R. King, R. Baraniuk and R. riedi, "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP", In Proc. INFOCOM 2005.

[BAINF01] Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms", Proc INFOCOM 2001.

[CTCPP06] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks", in 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006, Nara, Japan.

[CTCPT]   K. Tan, J. Song, M. Sridharan, and C.Y. Ho, "CTCP: Improving TCP-Friendliness Over Low-Buffered Network Links", Microsoft Technical Report.

[CUBIC]   I. Rhee, L. Xu and S. Ha, "CUBIC for fast long distance networks", Internet Draft, Expires Aug 31, 2007, draft-rhee-tcp-cubic-00.txt

[FAST]   C. Jin, D. Wei, S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance", in IEEE Infocom 2004.

[MSWRK]   Microsoft High-Speed TCP Summit, http://research.microsoft.com/events/TCPSummit/

[PADHYE]   J. Padhya, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", in Proc. ACM SIGCOMM 1998.

[RFC2988] V. Paxon and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3649] S. Floyd, "HighSpeed TCP for Large Congestion
           Windows", RFC 3649, Dec 2003.

[SLAC]   Yee-Ting Li, "Evaluation of TCP Congestion Control Algorithms on the Windows Vista Platform", SLAC-TN-06-005, http://www.slac.stanford.edu/pubs/slactns/tn04/slac-tn-06-005.pdf

[VEGAS]   L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", in Proc. ACM SIGCOMM, 1994.

Author's Addresses

Murari Sridharan
Microsoft Corporation
1 Microsoft Way, Redmond 98052

Email: muraris@microsoft.com

Kun Tan
Microsoft Research
5/F, Beijing Sigma Center
No.49, Zhichun Road, Hai Dian District
Beijing China 100080

Email: kuntan@microsoft.com

Deepak Bansal
Microsoft Corporation
1 Microsoft Way, Redmond 98052

Email: dbansal@microsoft.com

Dave Thaler
Microsoft Corporation
1 Microsoft Way, Redmond 98052

Email: dthaler@microsoft.com

Intellectual Property Statement

Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.