Network Coded TCP (CTCP) Performance over Satellite Networks

Jason Cloud and Muriel Médard
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, MA USA
email: {jcloud,medard}@mit.edu

Douglas Leith
Hamilton Institute
National University of Ireland Maynooth
Co. Kildare, Ireland
email: doug.leith@nuim.ie

***Abstract*—We show preliminary results for the performance of Network Coded TCP (CTCP) over large latency networks. While CTCP performs very well in networks with relatively short RTT, the slow-start mechanism currently employed does not adequately fill the available bandwidth when the RTT is large. Regardless, we show that CTCP still outperforms current TCP variants (i.e., Cubic TCP and Hybla TCP) for high packet loss rates (e.g., $> 2.5\%$). We then explore the possibility of a modified congestion control mechanism based off of H-TCP that opens the congestion window quickly to overcome the challenges of large latency networks. Preliminary results are provided that show the combination of network coding with an appropriate congestion control algorithm can provide gains on the order of 20 times that of existing TCP variants. Finally, we provide a discussion of the future work needed to increase CTCP's performance in these networks.**

Keywords*—*Network Coding; TCP; High Delay.

I. INTRODUCTION

It is widely known that TCP performs poorly over satellite networks [1], [2]. The combination of long round-trip times (RTT) and high packet loss rates (PER) over these networks create an environment that seriously degrades the performance of TCP. To overcome the challenges presented by satellite communication, a large variety of solutions have been proposed over the years. These range from modifications to TCP's congestion control algorithm to implementing performance enhancing proxies (PEPs). Each, of which, usually have their own drawbacks. In the case of modified TCP protocols, adoption is prevented due to the specialized nature of the protocol and issues related to fairness with other TCP variants. In the case of PEPs, increased hardware costs and issues regarding end-to-end semantics is an issue. In this paper, we suggest the use of Coded TCP (CTCP) proposed by Kim *et. al.*, [3], to overcome a large number of these issues.

Providing reliable data transport for satellite environments has been a topic of study since the late 1990's [1], [2]. End-to-end solutions typically involve tuning TCP so that the long RTTs representative of satellite links do not negatively impact performance. Two versions that perform well over satellite networks are TCP Cubic [4] and TCP Hybla [5]. Cubic, designed for high speed networks, and Hybla, designed for heterogeneous networks, use a congestion window algorithm that increases the congestion window size ($cwnd$) independently from the RTT. This makes either version useful in environments with high delay. Unlike TCP Cubic, Hybla was developed to also reduce the impact of multiple losses, inappropriate timeouts, and burstiness making it a more logical option for use over satellite links. When compared with each other, studies have shown that Hybla performs better than Cubic under high PERs while the reverse is true under low PER's [6]. Regardless, both experience performance degradation under high losses. A more recent protocol, Loss-Tolerant TCP (LT-TCP) [7], [8], combines Reed-Solomon (RS) coding with TCP to overcome this issue, but it requires the use of explicit congestion control (ECN) and the RS code can result in performance loss due to decoding errors. CTCP circumvents these issues by using a congestion control algorithm that does not rely on feedback from lower layers and network coding eliminates the possibility of decoding errors while helping to overcome packet losses.

In lieu of changes to TCP, PEPs are another common method used to increase performance over satellite links. A TCP flow is generally terminated at the gateway to the satellite link, a protocol specifically designed for the satellite system (usually one that is proprietary) is used to transmit data over the satellite network, and a new TCP session is setup on the other side of the satellite link to complete the connection. This implementation poses two issues. First, the cost of implementing a PEP at the satellite network gateway may be high. Second, the termination of TCP sessions at the PEP violates end-to-end semantics such as IPSEC [2]. Again, CTCP has the potential to eliminate the need for PEPs while providing the same level of service over satellite links.

In this paper, we explore CTCP's performance in high RTT environments to determine if we can provide resilience in the presence of high packet loss rates, in addition to achieving the performance of TCP Cubic or TCP Hybla under no packet losses. We first provide a description in Section II of the existing, well tested version of CTCP that uses a TCP Reno style slow-start mechanism. This version is designed to provide robustness to packet losses through the use of network coding, but the congestion window management is ill-suited to large bandwidth-delay products (BDP). In Section III we measure the performance of a modified version of CTCP that opens $cwnd$ in a manner similar to H-TCP [9] to show that it is indeed possible to achieve high performance with large PER's and RTT's. Finally, we conclude in Section IV by proposing areas of possible future work.

II. CTCP OVERVIEW

The development of CTCP [3] has shown how the integration of network coding with TCP can provide significant

 ISBN: 978-1-61208-317-9

benefits over existing TCP variants, especially in high packet loss environments. These gains are a direct result of the combination of both network coding and CTCP's congestion window management. The remainder of this section will provide a brief introduction into both of these mechanisms. For a more detailed explanation of CTCP's implementation and performance, the reader should refer to [3].

A. Network Coding

One of the key features of CTCP is the use of network coding to aid in recovery from packet losses and the capability to decrease overhead by limiting the number of required retransmissions. The gains provided by network coding are twofold: network coded packets can be used to provide forward error correction in the case of lost packets, and also simplifies feedback and retransmissions (should they be needed). In its current implementation, CTCP uses a systematic random linear code [10]. As an example, consider the transfer of packets $p_1 \dots p_k$ between a server and client. Each packet $p_i, i \in \{1, \dots, k\}$ is first sent uncoded followed by a number of network coded packets where every coded packet c_i is a random linear combination of the packets $p_1, \dots, p_k$, i.e.,

$$c_i = \sum_{j=1}^{k} \alpha_j p_j, \tag{1}$$

each $\alpha_j \in \mathbb{F}_{2^q}$ is randomly chosen, and q is large enough to ensure linear independence among all network coded packets with high probability (the current implementation draws α_j from $\mathbb{F}_{256}$). Should retransmissions be needed, additional network coded packets are generated and sent to the client. The number of coded packets sent along with the uncoded packets is dynamically determined based on an estimate of the path's packet loss probability, while the number of packets sent as a result of feedback is determined by both the number degrees of freedom ($dofs$) required by the client to decode and the estimated packet loss probability.

B. Congestion Control

The second feature of CTCP that is a major contributor to the observed gains is the congestion window management. CTCP uses a modified version of TCP's Additive Increase, Multiplicative Decrease (AIMD) algorithm that was designed to be compatible with network coding. Specifically, the current implementation modifies the multiplicative back-off factor, β, to be

$$\beta = \frac{RTT_{\min}}{RTT}, \tag{2}$$

where $RTT_{\min}$ is the path's estimated true round-trip propagation delay (which is assumed to be the lowest per-packet RTT observed during the lifetime of a connection) and RTT is the last measured round-trip time. The congestion window is increased using TCP Reno's increase mechanism (i.e., the slow-start mode increases $cwnd$ by 1 for every received acknowledgement, otherwise $cwnd$ is increased by $1/cwnd$).

This approach, in effect, assumes that the increase of a packet's RTT is solely due to the queuing of packets along the path, which is an indication of congestion. If a packet is lost at random and $RTT = RTT_{\min}$ (i.e., it is not lost due to congestion), then $cwnd$ is not reduced. On the other hand if $RTT > RTT_{\min}$, a packet loss is interpreted as congestion and $cwnd$ is reduced by a factor of β. While this approach does a fairly good job at distinguishing between packet losses due to congestion and poor channels, it has a few characteristics that may not work well for satellite networks. A more detailed discussion is provided in Section IV.

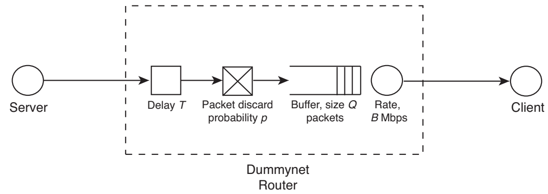


Figure 1: Schematic of experimental testbed.

C. Performance over Short RTT Networks

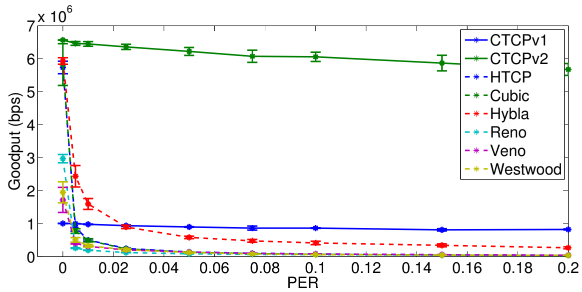
Using the mechanisms summarized above, [3] implemented CTCP as a SOCKSv5 proxy in user space and measured its performance over a wide range of conditions (although all measurements were made using round-trip times representative of terrestrial networks). Kim *et. al.* showed that CTCP can achieve goodput efficiencies greater than 90% for packet loss rates less than 0.2 while the performance of standard TCP variants is severely impacted. Another important aspect of CTCP is that it is friendly/fair with standard TCP, unlike some TCP variants that work well over satellite networks but are unfriendly to other TCP variants (e.g., TCP Hybla and Cubic TCP [11]). This is important since we are interested in providing an end-to-end solution. Therefore, we would like to ensure that if the bottleneck link is not the satellite link, CTCP does not adversely impact the performance of TCP flows not traversing the satellite.

III. CTCP Performance in Satellite Networks

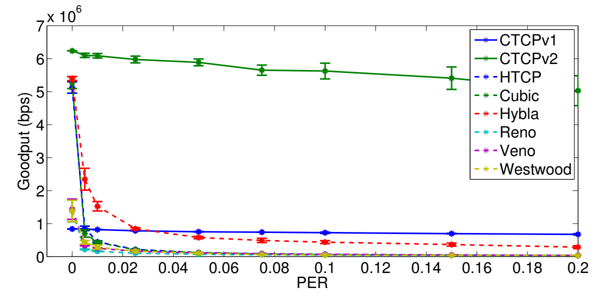
While previous results show that CTCP has great potential in networks with high packet losses and low RTT, no measurements exist for networks with large RTT. This section will explore the potential for CTCP to provide improved performance in environments with large delays using a testbed located at the Hamilton Institute, NUI Maynooth, Ireland.

The testbed used to collect measurements consists of commodity servers (Dell Poweredge 850, 3GHz Xeon, Intel 82571EB Gigabit NIC) connected via a router and gigabit switches. A diagram of the setup is shown in Figure 1. Sender and receiver machines used in the tests both run a Linux 2.6.32.27 kernel. The router is a commodity server running FreeBSD 4.11 and `ipfw-dummynet`. Data is transferred between the sender and receiver machines using `rsync` (version 3.0.4) and the appropriate TCP version.

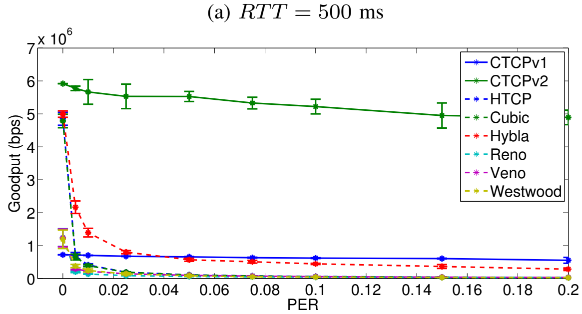
Each version of TCP, other than CTCP, is implemented within the kernel making it easy to select the appropriate variant. In the case of CTCP, it is implemented in user space as a SOCKSv5 proxy with the forward proxy located on the client and the reverse proxy located on the server. Traffic between the proxies is sent using CTCP. Therefore, a client's request is first directed to the local forward proxy, transmitted to the reverse proxy, and then forwarded to the appropriate port on



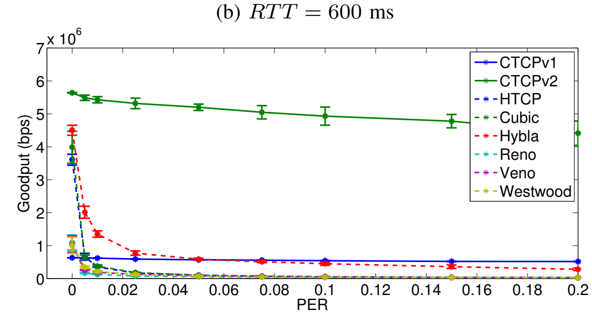
(a) $RTT = 500$ ms



(b) $RTT = 600$ ms



(c) $RTT = 700$ ms



(d) $RTT = 800$ ms

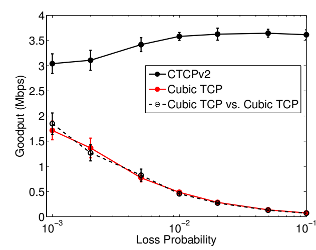
Figure 2: Comparison of TCP variants with varying PER and RTT with a link rate of 10 Mbps. Each bar shows the mean goodput, while the error bars show one standard deviation.

the server. The server responds using the reverse process. In order to use `rsync`, `proxychains` (version 3.1) was used to direct traffic to the proxy.

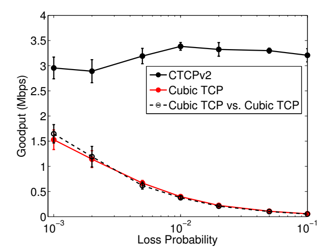
A series of tests were conducted using the following TCP variants: CTCP, Cubic, Hybla, Reno, Veno, and Westwood. A 20 MB file download is used, along with varying PER's and RTT's, to characterize the performance of each TCP version. Figure 2 provides a summary of each version's mean goodput as a function of the PER. Each test was run a minimum of three times and a maximum of ten times depending on the amount of time need to complete the 20 MB download.

The performance of CTCP, labeled "CTCP v1" in Figure 2, in networks with large RTT and low PER is significantly poorer than two of the TCP variants designed for these network types (i.e., TCP Hybla and TCP Cubic). For PER greater than 2.5%, CTCP begins to outperform both of these TCP variants for most of the RTTs tested. In fact, the goodput of CTCP remains relatively constant as the PER increases to 20% while the goodput of the other TCP variants approaches zero quickly.

The additive increase portion of CTCP's current congestion control algorithm is the primary reason for its poor performance at low PER. To overcome the challenges related to long RTT's, a modified version of CTCP, labeled "CTCP v2" in Figure 2, was implemented that increases $cwnd$ in a manner consistent with H-TCP (see [9] for more details). Because $cwnd$ is no longer dependent on the RTT, it can increase rapidly allowing it to use the available capacity more efficiently. In addition to the use of network coding and the unmodified multiplicative $cwnd$ back-off approach, CTCP is able to maintain a large throughput for PER's as high as 20%. In fact, measurements indicate that this modified version of CTCP provides a gain of approximately 21 times that of TCP



(a) $RTT = 500$ ms



(b) $RTT = 800$ ms

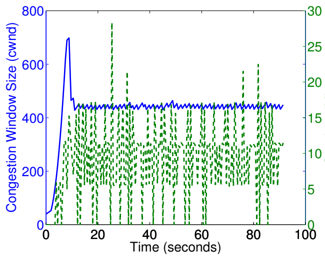
Figure 3: Goodput versus packet loss rate for (i) a Cubic TCP flow and CTCPv2 flow sharing a link (solid lines), and (ii) two Cubic TCP flows sharing a link (dashed line). The link rate is 5 Mbps and the error bars show one standard deviation.

Hybla for a PER of 20% and RTT of 500 ms over a link with a bandwidth of 10 Mbps.

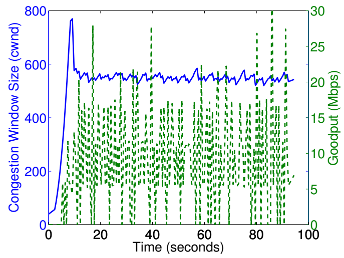
In addition, preliminary testing has shown that this version of CTCP is friendly with existing TCP versions. Each subfigure in Figure 3 provides a comparison of the throughput for two tests. The dotted line labeled "Cubic TCP vs. Cubic TCP" shows the throughput obtained by one Cubic TCP flow competing against a second Cubic TCP flow. The solid lines show the second test where a Cubic TCP flow is competing against a CTCP flow. The indication of fairness is provided by the similarity of the solid Cubic TCP line and the dotted line.

While these results are promising, additional work is required. This is evident in the trace of the goodput and $cwnd$ shown in Figure 4. The instantaneous goodput is highly variable, which causes delay jitter as packets are delivered to higher layers. Possible causes of this may be an underestimate of the packet loss probability or an underestimate of the number of coded packets needed. Either case creates the distinct decode events shown in the figure. Regardless, the

 ISBN: 978-1-61208-317-9



(a) $PER = 0.5\%$



(b) $PER = 20\%$

Figure 4: Trace of CTCPv2 over a 10 Mbps link with a RTT of 500 ms. The dotted line shows the the instantaneous goodput and the solid line shows $cwnd$. The mean goodput in (a) and (b) is 9.19 Mbps and 8.92 Mbps respectively.

potential for greatly increasing performance at the transport layer is evident. Not only can throughput performance be drastically improved, but this solution appears, at first glance, to be backward compatible with existing TCP variants.

IV. Conclusions and Future Work

In Section II, an overview of CTCP is presented and a brief overview of CTCP's performance in networks with short RTT was provided. In Section III, CTCP's performance is compared with other TCP variants in networks with RTT's similar to those that would be observed with satellite communications. Measurements showed that the current implementation of CTCP performed worse than some existing TCP versions for small PER, but outperformed other TCP versions for high PER. One of the primary causes of this was discussed and an alternate $cwnd$ increase mechanism is used to highlight that some minor changes to the current congestion control algorithm can significantly improve performance in these environments. While previous sections discussed several areas of future research such as this, the remainder of this section will introduce additional future research directions.

Additional research into CTCP's congestion control algorithm is required. First, the current implementation relies on the RTT of the path in order to increase $cwnd$. For connections with large RTT's, this is obviously an issue. More aggressive methods for increasing $cwnd$, such as using a H-TCP like mechanism, need to be thoroughly developed while still maintaining interoperability with network coding and fairness with legacy TCP variants. Second, the method for determining whether a packet is lost due to congestion or due to a poor link is also an issue. The multiplicative back off method currently used (i.e., $\beta = {}^{RTT_{\min}}/_{RTT}$) works well when jitter in the RTT measurement is primarily caused by the filling of queues. It fails to work properly when the delay jitter is caused by something else. For example, if the delay jitter is caused by a particular medium-access (MAC) method, it is likely that $\beta < 1$ causing $cwnd$ to collapse. This was observed in measurements taken over a WiMax network [3] where the MAC's scheduling algorithm caused large variations in the RTT. Methods using feedback from the network, such as LT-TCP, which uses explicit congestion notification (ECN) [7], are possibilities although we would like to ensure that CTCP operates irrespective of lower layer implementations.

The use of network coding in CTCP is critical for overcoming packet losses and providing high throughput, but little is understood about how to adjust the number of packets coded together (i.e., the coding window). The current implementation of CTCP uses a fixed size block, or generation, scheme for generating network coded packets. This is not optimal given the user's requirements since there is an inherent tradeoff between throughput and delay as the block, or generation, size is changed [12]. Dynamically adjusting the block size, or using a sliding coding window approach, to meet the user's throughput/delay requirements is also a topic of ongoing research. Furthermore, the interaction between congestion avoidance and network coding is not fully understood. The current implementation treats both congestion avoidance and network coding separately, yet there is evidence that intelligently merging the two can provide a performance increase.

In summary, CTCP has potential to greatly improve network performance over existing transport layer protocols in the presence of both high packet error rates and round-trip times. Initial measurements have shown significant gains in goodput over existing TCP versions, but additional research is needed to tune both the congestion control algorithm and network coding parameters to ensure proper functionality.

Acknowledgment

This work is sponsored, in part, by the Assistant Secretary of Defense (ASD R&E) under Air Force Contract # FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

References

[1] C. Caini, R. Firrincieli, M. Marchese, T. d. Cola, M. Luglio, C. Roseti, N. Celandroni, and F. Potorti, "Transport Layer Protocols and Architectures for Satellite Networks," International Journal of Satellite Communications and Networking, vol. 25, no. 1, 2007, pp. 1–26.

[2] A. Pirovano and F. Garcia, "A New Survey on Improving TCP Performances Over Geostationary Satellite Link," Network and Communication Technologies, vol. 2, no. 1, Jan. 2013.

[3] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Médard, "Network Coded TCP (CTCP)," CoRR, vol. abs/1212.2291v2, 2012. [Online]. Available: http://arxiv.org/abs/1212.2291

[4] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," SIGOPS Oper. Syst. Rev., vol. 42, no. 5, Jul. 2008, pp. 64–74.

[5] C. Caini and R. Firrincieli, "TCP Hybla: A TCP Enhancement for Heterogeneous Networks," International Journal of Satellite Communications and Networking, vol. 22, no. 5, 2004, pp. 547–566.

[6] S. Trivedi, S. Jaiswal, R. Kumar, and S. Rao, "Comparative Performance Evaluation of TCP Hybla and TCP Cubic for Satellite Communication Under Low Error Conditions," in IMSAA, 2010, pp. 1–5.

[7] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle, "Loss-Tolerant TCP (LT-TCP): Implementation and Experimental Evaluation," in MILCOM, 2012, pp. 1–6.

[8] V. Sharma, S. Kalyanaraman, K. Kar, K. K. Ramakrishnan, and V. Subramanian, "MPLOT: A Transport Protocol Exploiting Multipath Diversity Using Erasure Codes, year = 2008," in INFOCOM, pp. 121–125.

[9] D. Leith and R. Shorten, "H-TCP: TCP for High-Speed and Long-Distance Networks," 2004.

[10] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," IEEE Transactions on Information Theory, vol. 52, no. 10, 2006, pp. 4413–4430.

[11] A. Urke, L. Braten, and K. Ovsthus, "TCP Challenges in Hybrid Military Satellite Networks; Measurements and Comparison," in MILCOM, 2012, pp. 1–6.

[12] W. Zeng, C. Ng, and M. Médard, "Joint Coding and Scheduling Optimization in Wireless Systems with Varying Delay Sensitivities," in SECON, 2012, pp. 416–424.