Congestion Control for Coded Transport Layers

MinJi Kim*, Jason Cloud*, Ali ParandehGheibi*, Leonardo Urbina*,
Kerim Fouli*, Douglas J. Leith†, and Muriel Médard*

***Abstract*—The application of congestion control can have a significant detriment to the quality of service experienced at higher layers, especially under high packet loss rates. The effects of throughput loss due to the congestion control misinterpreting packet losses in poor channels is further compounded for applications such as HTTP and video leading to a significant decrease in the user's quality of service. Therefore, we consider the application of congestion control to transport layer packet streams that use error-correction coding in order to recover from packet losses. We introduce a modified AIMD approach, develop an approximate mathematic model suited to performance analysis, and present extensive experimental measurements in both the lab and the "wild" to evaluate performance. Our measurements highlight the potential for remarkable performance gains, in terms of throughput and upper layer quality of service, when using coded transports.**

I. Introduction

We consider congestion control for transport layer packet streams that use error-correction coding to recover from packet loss. Recently, there has been a resurgence of interest in the use of coding at the transport layer [1]–[5]. Much of this has been driven by the ubiquity of wireless connectivity at the network edge. Growing trends toward cellular traffic offloading onto 802.11 wireless links and the increasing density of wireless deployments is making interference a major contributor to packet erasures in unlicensed and white-space bands. As a result, the quality of wireless links at the network edge is becoming much more diverse and challenging.

Addressing these issues at the transport layer is very appealing. Unlike link layer changes, transport layer solutions help to ensure backward compatibility and enables mass deployment. For example, a recent industry study [6] estimated that almost 1.2 billion 802.11 devices have been shipped to date. Replacing these devices, in addition to other existing wireless infrastructures, to incorporate new link layer technologies is largely impractical due to the costs involved.

A key issue with transport layer solutions is congestion control, which is responsible for protecting the network from congestion collapse, ensuring reasonably efficient use of available capacity, and allocating capacity in a roughly fair manner. When links experience packet losses due to poor channels or interference, lost packets are no longer synonymous with network congestion. Furthermore if error-correction coding is used to recover from packet losses, packet retransmissions become unnecessary and the trade-off between throughput and delay changes. In addition, notions of fairness need to be revisited because different paths may have different levels of packet losses (*e.g.* should a lossy path be given more or less bandwidth than a loss-free path sharing the same bottleneck?).

In this paper, we introduce a modified additive increase, multiplicative decrease (AIMD) approach to congestion control for coded packet flows that helps to meet the following goals:

1) Provide high throughput under independent and identically distributed (i.i.d.) and correlated packet losses. These losses may occur due to a) channels with high interference or fading, b) hidden terminals in wireless networks, and c) any additional known or unknown cause of packet loss not related to congestion.
2) Operate under a wide range of network conditions (*i.e.*, different round-trip times (RTT) and packet loss rates).
3) Be friendly to other non-coded flows.
4) Provide a better quality of service for higher layer applications than current transport protocols.

We also develop an approximate mathematic model for performance analysis, implement it using a protocol called Network Coded TCP (CTCP) ([7] and [8] provide detailed descriptions of the implementation), and present extensive experimental measurements. Our measurements highlight the potential for remarkable performance gains when using coded transports. In controlled lab experiments, with i.i.d. packet losses, we find reductions of more than an order of magnitude (*i.e.* >1000%) in completion times for both HTTP and streaming video flows when the link packet loss rate exceeds 5%. These gains translate into significant increases in the quality of service provided by upper layer applications (*e.g.*, the number of buffer under-runs experienced during the playback of a video drops from approximately 50 to zero). Note that these benefits take into account encoding/decoding delay associated with coded transport protocols, indicating that the cost of encoding/decoding is negligible given the immense gain in throughput. Measurements are also taken in real networks (*i.e.*, in the "wild" at public WiFi hotspots) where the packet losses are not necessarily i.i.d. or known. These measurements also show reductions in connection completion times of 100-300% compared with standard TCP, which does not use error-correction coding. Importantly, these gains do not come at the cost of penalizing standard TCP flows sharing the same links.

II. Preliminaries

Reference [1] first introduced TCP/NC, which inserts a network coding layer between TCP and IP in the protocol stack. The network coding layer intercepts and modifies TCP's acknowledgement scheme such that random erasures do not affect the transport layer's performance. Our work builds upon

*M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, and M. Médard are with the Massachusetts Institute of Technology, MA USA (e-mail: {minjikim, jcloud, parandeh, lurbina, fouli, medard}@mit.edu).

†D. Leith is with the Hamilton Institute, NUI Maynooth, Ireland (e-mail: doug.leith@nuim.ie).

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.

TCP/NC using a new coded transport protocol, CTCP, that is more efficient, robust, and adaptive than its predecessor.

Congestion control operation is logically decoupled from the specific coding scheme employed since congestion control relates to the scheduling of packet transmissions while the coding scheme relates to the contents of the packets transmitted (*e.g.* coded or uncoded). As discussed in [7] and [8], this decoupling can be implemented in practice by using *tokens* to control the sender's transmission rate instead of the congestion window $cwnd$. Therefore, $tokens$ play a similar role for coded TCP as $cwnd$ does for TCP. If a $token$ is available, the sender can transmit a single packet (either coded or uncoded). Once the packet is sent, the $token$ is used. New $tokens$ are generated upon reception of feedback indicating the packet is no longer on the fly.

Our focus here is on congestion control; but in order to carry out the experimental performance evaluation, we also need to specify the coding scheme. Information packets are queued at the sender, but also transmitted in order without delay. After a block of $N = 32$ packets are transmitted, $N_c = N(\frac{1}{1-p} - 1)$ coded packets are inserted into the packet stream to help recover from lost packets where p is an estimate of the path's packet loss rate based on ACK feedback from the receiver. Coded packets are constructed using an equiprobable random linear network code in field $GF(2^8)$ [9]. If the receiver is unable to decode the block after the first round of transmissions, additional coded packets are transmitted upon receipt of the appropriate feedback. Implementation details and additional discussion concerning CTCP are provided in [7] and [8].

III. Congestion Control

Traditional TCP's AIMD congestion control increases the sender's congestion window size $cwnd$ by α packets per RTT and multiplicatively decreases $cwnd$ by a backoff factor β on detecting packet losses. The typical values are $\alpha = 1$ when appropriate byte counting is used, and $\beta = 0.5$. On lossy links, repeated backoffs in response to noise, rather than queue overflow, can prevent $cwnd$ from increasing to fill the available capacity. The behavior is well known and is captured. For example, $cwnd$ scales as $\sqrt{1.5/p}$ in [10], where p is the packet loss rate.

On lossy links, packet loss is not a reliable indicator of network congestion. One option might be to use delay, rather than loss, as the indicator of congestion; but this raises many new issues and purely delay-based congestion control approaches have not been widely adopted in the internet despite being the subject of extensive study. Another option might be to use explicit signaling, (*e.g.* via ECN). However, this requires both network-wide changes and disabling $cwnd$ backoffs when packets are lost. These considerations motivate consideration of hybrid approaches, making use of both loss and delay information. The use of hybrid approaches is well-established, for example Compound TCP [11] is widely deployed.

We consider modifying the AIMD multiplicative backoff to

$$\beta = \frac{RTT_{min}}{RTT}, \tag{1}$$

where RTT_{min} is the path round-trip propagation delay (typically estimated as the lowest per packet RTT observed during the lifetime of a connection) and RTT is the current round-trip time. The choice for β in Eq. (1) decreases the flow's $tokens$ so that the link queue just empties and full throughput is maintained upon packet loss. This is similar to the approach considered in [12], which uses $\beta = RTT_{min}/RTT_{max}$. In fact, Eq. (1) reduces to the approach in [12] on links with only queue overflow losses since $RTT = RTT_{max}$ (the link queue is full) when loss occurs. If a link is provisioned with a bandwidth-delay product of buffering, as per standard guidelines, then $RTT_{max} = 2RTT_{min}$ and $\beta = 0.5$ (*i.e.,* the behavior is identical to that of standard TCP). More generally, the sum of n flows' throughput must equal the link capacity B (*i.e.,* $\sum_{i=1}^{n} tokens_i/RTT_i = B$) when a queue overflow occurs. After a packet loss is identified, backoff occurs according to Eq. (1), and the sum-throughput becomes $\sum_{i=1}^{n} \beta_i tokens_i/RTT_{min,i} = B$ allowing the queue to empty.

When a network path is under-utilized and a packet loss occurs, $RTT = RTT_{min}$ resulting in $\beta = 1$ and $\beta \times tokens = tokens$ (*i.e.,* $tokens$ is not decreased). Hence, $tokens$ is able to grow despite the presence of packet losses. Once the link starts to experience queueing delay, $RTT > RTT_{min}$ making $\beta < 1$ (*i.e.,* $tokens$ is decreased on loss). Since the link queue is filling, the sum-throughput before loss is $\sum_{i=1}^{n} tokens_i/RTT_i = B$. After each flow decreases their $tokens$ in response to queue overflow losses, the sum-throughput is at least $\sum_{i=1}^{n} \beta_i tokens_i/RTT_{min,i} = B$ when the queue empties (*i.e.,* Eq. (1) adapts β to maintain full throughput).

While we have focused primarily on modifications to the backoff factor β in combination with the standard linear additive increase (*i.e.,* where α is constant), we note that our adaptive backoff can be combined with other additive increase approaches. In particular, approaches similar to Cubic or Compound TCP can be substituted for the one used in this paper.

IV. Throughput Performance Modeling

Consider a link shared by n flows where we define B to be the link capacity and T_i to be the round-trip propagation delay of flow i. We will assume that the queueing delay is negligible (*i.e.,* the queues are small), and differences in the times when flows detect packet losses due to differences in propagation delays can be neglected. Let t_k denote the time of the k-th network backoff event, where a network backoff event is defined to occur when one or more flows reduce their $tokens$. Furthermore, let $w_i(k)$ denote the $tokens$ of flow i immediately before the k-th network backoff event and let $s_i(k) = w_i(k)/T_i$ be the corresponding throughput. We then have

$$s_i(k) = \tilde{\beta}_i(k-1)s_i(k-1) + \tilde{\alpha}_i T(k), \tag{2}$$

where $\tilde{\alpha}_i = \alpha/T_i^2$, α is the AIMD increase rate in packets per RTT, $T(k)$ is the time in seconds between the $k-1$ and k-th backoff events, and $\tilde{\beta}_i(k)$ is the backoff factor of flow i at event k. The backoff factor $\tilde{\beta}_i(k)$ is a random variable, which takes the value 1 when flow i does not experience a loss at network event k and takes the value given by Eq. (1) otherwise. The time $T(k)$ is also a random variable with a distribution dependent on the packet loss process and typically coupled to the flow rates $s_i(k)$, $i = 1, \cdots, n$.

For example, associate a random variable δ_j with packet j, where $\delta_j = 1$ when packet j is erased and 0 otherwise. Assume δ_j are i.i.d with erasure probability p. Then $Prob(T(k) \leq t) = 1 - (1-p)^{N_t(k)}$ where $N_t(k) = \sum_{i=1}^{n} N_{f,i}(t)$ is the total number of packets transmitted over the link in interval t following backoff event $k-1$ and $N_{t,i}(k) = \tilde{\beta}_i(k-1)s_i(k-1)t + 0.5\tilde{\alpha}_i t^2$ is the number of packets transmitted by flow i in this interval t. Also, the probability $\gamma_i(k) := Prob(\tilde{\beta_i(k)} = 1)$ that flow i does not back off at the k-th network backoff event is the probability that it does see any loss during the RTT interval $[T(k), T(k) + T_i]$. This can be approximated by $\gamma_i(k) = (1-p)^{s_i(k)T_i}$ on a link with sufficiently many flows.

Since both $\tilde{\beta}_i(k)$ and $T(k)$ are coupled to the flow rates $s_i(k)$, $i = 1, \ldots, n$, analysis of the network dynamics is generally challenging; although the analysis becomes fairly straightforward when the backoff factor $\tilde{\beta}_i(k)$ is stochastically independent of the flow rate $s_i(k)$. Note that this assumption is valid in a number of useful and interesting circumstances. One such circumstance is when links are loss-free (with only queue overflow losses) [13]. Another is on links with many flows and i.i.d packet losses where the contribution of a single flow i to the queue occupancy is small. Furthermore, experimental measurements provided later in the paper indicate that analysis using the independence assumption accurately predicts performance over a range of other network conditions suggesting that the results are insensitive to this assumption.

Given independence, the expected throughput using Eq. (2) is

$$\mathbb{E}[s_i(k)] = \mathbb{E}[\tilde{\beta}_i(k-1)]\mathbb{E}[s_i(k-1)] + \tilde{\alpha}_i \mathbb{E}[T(k)]. \quad (3)$$

When the network is also ergodic, a stationary distribution of flow rates exists. Let $\mathbb{E}[s_i]$ denote the mean stationary rate of flow i. From Eq. (3), we have

$$\mathbb{E}[s_i] = \frac{\tilde{\alpha}_i}{1 - \mathbb{E}[\tilde{\beta}_i]}\mathbb{E}[T]. \quad (4)$$

Since the factor $\mathbb{E}[T]$ is common to all flows, the fraction of link capacity obtained by flow i is determined by $\frac{\tilde{\alpha}_i}{1-\mathbb{E}[\tilde{\beta}_i]}$.

Fairness between flows with the same RTT: When flows i, j have the same RTT, $\tilde{\alpha}_i = \tilde{\alpha}_j$ from Eq. (4). Therefore, both flows have the same mean backoff factor $\mathbb{E}[\tilde{\beta}_i] = \mathbb{E}[\tilde{\beta}_j]$; and they obtain, on average, the same throughput share.

Fairness between flows with different RTTs: When flows i, j have different round trip times ($T_i \neq T_j$) but the same mean backoff factor, the ratio of their throughputs is $\frac{\mathbb{E}[s_i]}{\mathbb{E}[s_j]} = (\frac{T_j}{T_i})^2$. Observe that this is identical to standard TCP behavior [13].

Fairness between flows with different loss rates: The stationary mean backoff factor $\mathbb{E}[\tilde{\beta}_i]$ depends on the probability that flow i experiences a packet loss at a network backoff event. If two flows i and j experience different per packet loss rates p_i and p_j (*e.g.,* they might have different access links while sharing a common throughput bottleneck), fairness is affected through $\mathbb{E}[\tilde{\beta}_i]$.

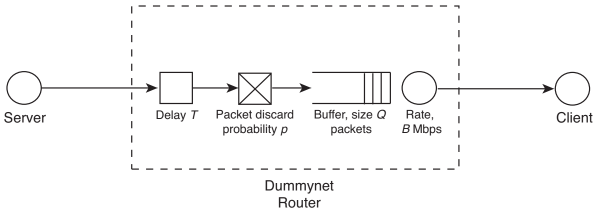


Fig. 1: Schematic of experimental testbed.

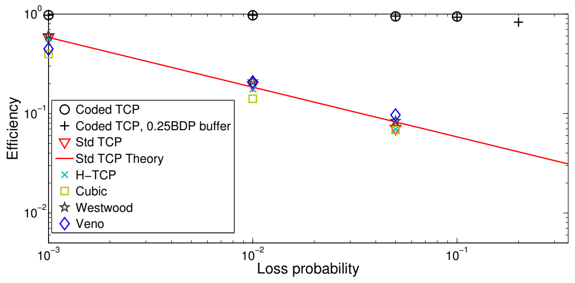
Friendliness: Eq. (2) is sufficiently general enough to include AIMD with a fixed backoff factor, which is used by standard TCP. We consider two cases. First, consider a loss-free link where the only losses are due to queue overflow and all flows backoff when the queue fills. Under this case, $\mathbb{E}[\tilde{\beta}_i] = \beta_i(k)$. For flow i with fixed backoff of 0.5 and flow j with adaptive backoff β_j, the ratio of the mean flow throughputs is $\mathbb{E}[s_i]/\mathbb{E}[s_j] = 2(1-\beta_j)$ (by Eq. (4)), assuming both flows have the same RTT. Note that the throughputs are equal when $\beta_j = T_j/RTT_j = 0.5$. Since $RTT_j = T_j + q_{max}/B$ where q_{max} is the link buffer size, $\beta_j = 0.5$ when $q_{max} = BT_j$ (*i.e.,* the buffer is half the size of the bandwidth-delay product). Second, consider the case when the link has i.i.d packet losses with probability p. If p is sufficiently large, the queue rarely fills and queue overflow losses are rare. The throughput of flow i with a fixed backoff of 0.5 can then be accurately modeled using the Padhye model [10]. Specifically, the throughput is largely decoupled from the behavior of other flows sharing the link, since coupling takes place via queue overflow. This means that flows using an adaptive backoff do not penalize flows that use a fixed backoff. Section V-C presents experimental measurements confirming this behavior.

V. Experimental Measurements

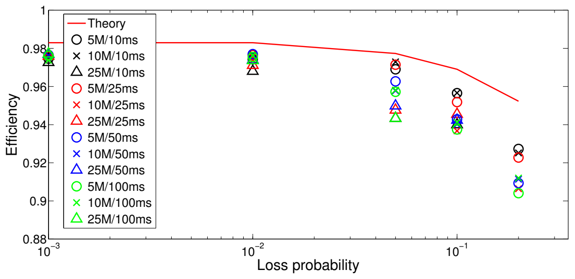
Experimental measurements, both in a controlled setting and in the "wild", were taken to determine the effectiveness of our proposed congestion control in meeting the goals listed in Section I. The throughput performance over a wide range of packet loss rates and RTT's were measured to verify performance is not degraded in these environments. Fairness with legacy transport protocols (*i.e.,* various TCP implementations) is measured with and without packet losses showing that our congestion control is fair with other non-coded flows. Furthermore, measurements were taken to show the benefits to the quality of user experience that our congestion control has on upper layer applications. Specifically, the completion time of HTTP requests and the number of buffer underruns experienced when downloading and playing a video are measured. Finally, a series of real-world measurements were taken to show that the congestion control is capable of not only handling i.i.d. packet losses, but also correlated packet losses.

A. Testbed Setup

The lab testbed consists of commodity servers (Dell Poweredge 850, 3GHz Xeon, Intel 82571EB Gigabit NIC) connected via a router and gigabit switches (Figure 1). Sender



(a) Link 25 Mbps, RTT 20 ms



(b) CTCP

Fig. 2: Measurements of goodput efficiency against packet loss rate, link rate and RTT. The Theory curve in Figure 2b is generated using Eq. (5).

and receiver machines used in the tests both run a Linux 2.6.32.27 kernel. The router is also a commodity server running FreeBSD 4.11 and `ipfw-dummynet`. It can be configured with various propagation delays T, packet loss rates p, queue sizes Q and link rates B to emulate a range of network conditions. As indicated in Figure 1, packet losses in `dummynet` occur before the rate constraint, not after. Therefore, packet losses do not reduce the bottleneck link capacity B. Unless otherwise stated, appropriate byte counting is enabled for standard TCP and experiments are run for at least 300 s. Data traffic is generated using `rsync` (version 3.0.4), HTTP traffic using `apache2` (version 2.2.8) and `wget` (version 1.10.2), and video traffic using `vlc` as the both server and client (version 0.8.6e as server, version 2.0.4 as client).

Coded TCP (CTCP) is implemented in userspace as a forward proxy located on the client and a reverse proxy located on the server. This has the advantage of portability and of requiring neither root-level access nor kernel changes. Traffic between the proxies is sent using CTCP. With this setup, a client request is first directed to the local forward proxy. This transmits the request to the reverse proxy, which then sends the request to the appropriate port on the server. The server response follows the reverse process. The proxies support the SOCKS protocol and standard tools allow traffic to be transparently redirected via the proxies. In our tests, we used `proxychains` (version 3.1) for this purpose.

B. *Efficiency*

Figure 2 presents experimental measurements of the efficiency (equal to $\frac{\text{goodput}}{\text{link capacity}}$) of various TCP implementations and CTCP over a range of network conditions. Figure 2a shows the measured efficiency versus the packet loss probability p for a 25 Mbps link with a 25 ms RTT and a bandwidth-delay product of buffering. Baseline data is shown for standard

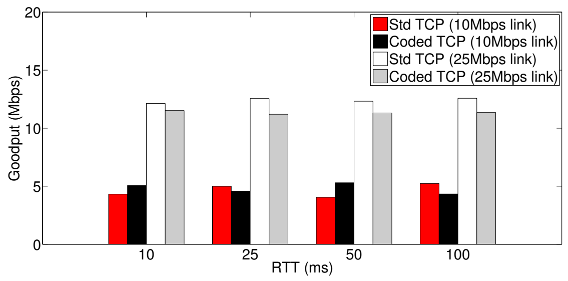
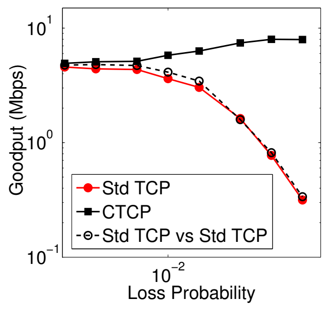
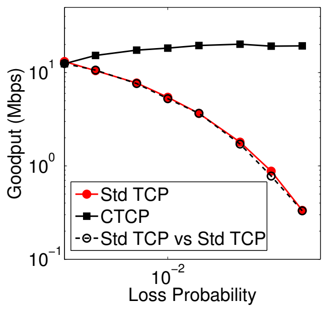


Fig. 3: Goodput for a standard TCP and a CTCP flow sharing a loss-free link; results are shown for 10 Mbps and 25 Mbps links with varying RTTs.



(a) Lossy 10 Mbps link with RTT = 25 ms



(b) Lossy 25 Mbps link with RTT = 25 ms

Fig. 4: Goodput against link loss rate for (i) a TCP and a CTCP flow sharing this link (solid lines), and (ii) two TCP flows sharing lossy link (dashed line).

TCP (*i.e.,* TCP SACK/Reno), Cubic TCP (current default on most Linux distributions), H-TCP, TCP Westwood, TCP Veno, together with the value $\sqrt{1.5/p}$ packets per RTT that is predicted by the popular Padhye model [10]. It can be seen that the standard TCP measurements are in good agreement with the Padhye model; and each of the non-CTCP implementations closely follow the standard TCP behavior because the link bandwidth-delay product of 52 packets lies in the regime where TCP variants seek to ensure backward compatibility with standard TCP. Furthermore, the achieved goodput falls to 20% of the link capacity when the packet loss rate is just 1%. This behavior of standard TCP is well known. However, the efficiency measurements for CTCP (provided in Figure 2a and also given in more detail in Figure 2b) show that the goodput is $> 96\%$ of link capacity for a loss rate of 1%. This is a five-fold increase in goodput compared to standard TCP.

Figure 2b further shows that CTCP is not sensitive to the link rate or RTT by presenting CTCP efficiency measurements for a range of link rates, RTTs, and loss rates. Also shown in Figure 2b is a theoretical upper bound on the efficiency calculated using

$$\eta = \frac{1}{N}\sum_{k=0}^{n-1}(n-k)\binom{n}{k}p^k(1-p)^{N-k}, \tag{5}$$

where $N = 32$ is the block size, p the packet erasure probability, and $n = \lfloor N/(1-p)\rfloor - N$ is the number of forward-transmitted coded packets sent with each block. The value η is the mean number of forward-transmitted coded packets that are unnecessary because there are fewer then n

erasures.

As discussed in Section III, the efficiency achieved by CTCP is also insensitive to the buffer provisioning. This property is illustrated in Figure 2a, which presents CTCP measurements when the link buffer is reduced in size to 25% of the bandwidth-delay product. The efficiency achieved with 25% buffering is close to that with a full bandwidth-delay product of buffering.

C. Friendliness with Standard TCP

Figures 3 and 4 confirm that standard TCP and CTCP can coexist in a well-behaved manner. In these measurements, a standard TCP flow and a CTCP flow share the same link competing for bandwidth. As a baseline, Figure 3 presents the goodputs of TCP and CTCP for range of RTTs and link rates on a *loss-free* link (*i.e.,* when queue overflow is the only source of packet loss). As expected, the standard TCP and CTCP flows consistently obtain similar goodput.

Figure 4 presents goodput data when the link is lossy. The solid lines indicate the goodputs achieved by CTCP and standard TCP sharing the same link with varying packet loss rates. At low loss rates, they obtain similar goodputs; but as the loss rate increases, the goodput of standard TCP rapidly decreases (as already observed in Figure 2a). It is also important to note that CTCP's goodput actually increases as the standard TCP goodput decreases. Because the standard TCP flow cannot fully use the available bandwidth, CTCP is able to use this bandwidth and obtain a higher goodput.

For comparison, Figure 4 also shows (using the dotted lines) the goodput achieved by a standard TCP flow when competing against another standard TCP flow (*i.e.,* when two standard TCP flows share the link). Note that the goodput achieved is close to that achieved when sharing the link with a CTCP flow. This demonstrates that CTCP does not penalize the standard TCP flow under both low and high packet loss rates.

D. Application Performance

The performance of a particular congestion control algorithm can have serious, non-linear, impacts to upper layer applications. While throughput is usually the primary measure used, the effects of congestion control on the upper layers is possibly even more important since they usually impact the quality of service or user experience. As a result, two upper layer applications (HTTP and streaming video) are used to measure the performance of our congestion control scheme.

1) Web: Figure 5 shows measurements of HTTP request completion time against file size for standard TCP and CTCP. The completion times with CTCP are largely insensitive to the packet loss rate. For larger file sizes, the completion times approach the best possible performance indicated by the dashed line. For smaller file sizes, the completion time is dominated by slow-start behavior. Note that CTCP and TCP achieve similar performance when the link is loss-free; however, TCP's completion time quickly increases with loss rate. For a 1 MB connection, the completion time with standard TCP increases from 0.9 s to 18.5 s as the loss rate increases from 1% to 20%. For a 10 MB connection the corresponding increase is from 7.1 s to 205 s. This is a reduction of more than 20× and 30× for a 1 MB and 10 MB connection, respectively.

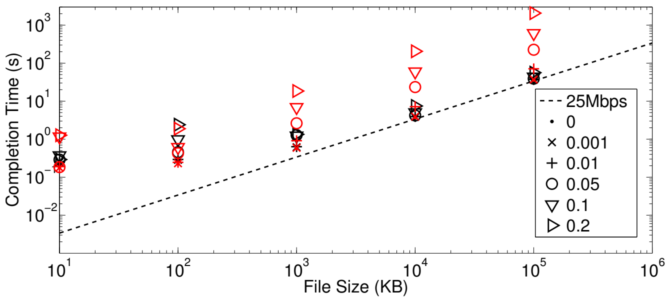
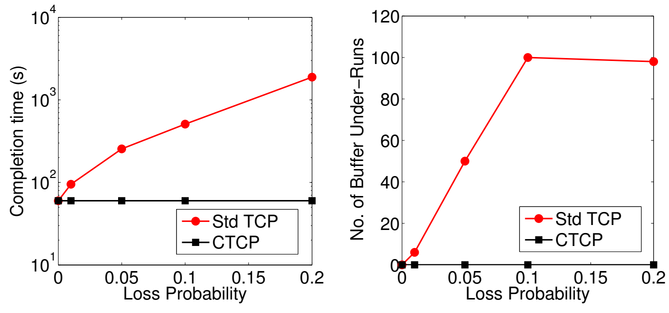


Fig. 5: Measured HTTP request mean completion time against file size over 25 Mbps link with RTT = 10 ms. Data is shown for standard TCP (red) and CTCP (black) for a range of loss rates. Error bars are comparable in size to the symbols used in the plot and so are omitted. Note the log scale on the y-axis.



(a) Completion Time

(b) Buffer Under-runs

Fig. 6: Measurements of video streaming performance against loss rate with a 25 Mbps link and a RTT of 10 ms. Data is shown for standard TCP and CTCP. Figure 6a shows the running time taken to play a video of nominal duration (60 s); Figure 6b shows the number of under-runs of the playout buffer at the client.

2) Streaming Video: Figure 6 plots performance measurements for streaming video for a range of packet loss rates on a 25 Mbps link with RTT equal to 10 ms. A `vlc` server and client are used to stream a 60 s video. Figure 6a plots the measured time for playout of the video to complete. Again, note the log scale on the y-axis.

The playout time with CTCP is approximately 60 s and is insensitive to the packet loss rate. In contrast, the playout time with standard TCP increases from 60 s to 95 s when the loss rate is increased from 0% to 1%. It increases further to 1886 s (31 minutes) as the loss rate is increased to 20% (more than 30× slower than when using CTCP).

Figure 6b plots measurements of the playout buffer under-run events at the video client. No buffer under-run events occur when using CTCP even when the loss rate is as high as 20%. With standard TCP, the number of buffer under-runs increases with loss rate until it reaches a plateau at around 100 events, corresponding to a buffer underrun occurring after every playout of a block of frames. In terms of user experience, the increases in running time result in the video repeatedly stalling for long periods of time and are indicative of a thoroughly unsatisfactory quality of experience, even at a loss rate of 1%.

These benefits also take into account the encoding/decoding delay associated with CTCP indicating that the cost of encod-

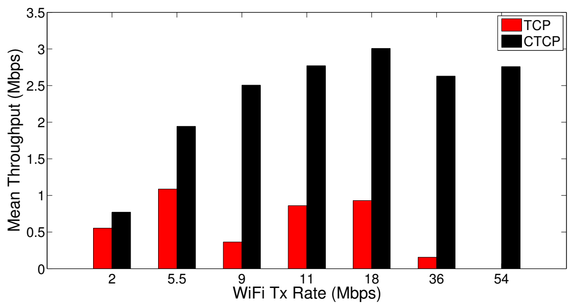


Fig. 7: Measurements of mean throughput vs wireless PHY rate used with standard TCP (Cubic TCP) and CTCP on an 802.11 link affected by microwave oven interference.

ing/decoding is negligible given the immense gain in throughput. For example, the CTCP receiver may need to receive N coded packets before it can even decode the first packet among the N coded packets. However, given the significant gain in throughput, the time needed to receive and decode N packets become negligible. In addition, the use of systematic network coding (as explained in [7] and [8] helps to minimize the potential delay associated with encoding/decoding.

VI. Real-world Performance

While the measurements shown in the previous section provide insight into the performance of our congestion control within a controlled setting using fixed RTTs and i.i.d. packet losses, it is necessary to verify its performance in real-world networks where packet losses are not i.i.d. Unless otherwise stated, the default operating system settings are used for all network parameters.

A. Microwave Oven Interference

We consider a 802.11b/g wireless client downloading a 50 MB file from an access point (AP) over a link subject to interference from a microwave oven (MWO). The wireless client and AP were equipped with Atheros 802.11b/g 5212 chipsets (radio 4.6, MAC 5.9, PHY 4.3 using Linux MadWifi driver version 0.9.4) operating on channel 8. The MWO used was a 700 W Tesco MM08 17L, which operates in the 2.4 GHz ISM band with significant overlap ($> 50\%$) with the WiFi 20 MHz channels 6 to 13. Its interference is approximately periodic with a period of 20 ms (*i.e.,* 50 Hz) and mean pulse width of 9 ms (the width was observed to fluctuate due to frequency instability of the MWO cavity magnetron, a known effect in MWOs).

Figure 7 presents measurements of the mean throughout achieved over the file download versus the PHY rate used on the downlink. Data is shown using standard TCP (in this case Cubic TCP) and CTCP. Data for a PHY rate of 1 Mbps is not shown because the packet loss rate was close to 100% due to the 1500 B frame's transmission time being greater than the interval between the MWO's interference bursts. It can be seen that the throughput achieved by standard TCP rises slightly as the PHY rate is increased from 1 Mbps to 5.5 Mbps, but then decreases to zero for PHY rates above 36 Mbps. In comparison, CTCP's throughput is approximately double (*i.e.,* 200%) that of standard TCP at a PHY rate of 5.5 Mbps, more

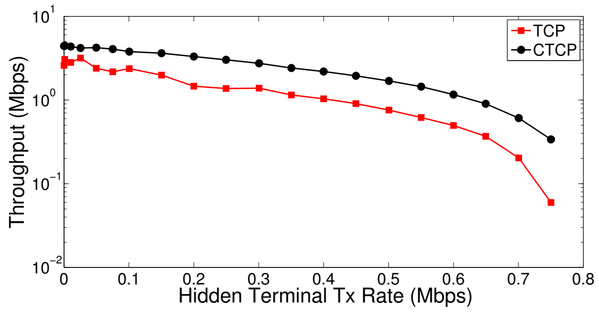


Fig. 8: Throughput vs intensity of hidden terminal interference when using standard TCP (Cubic TCP) and CTCP over an 802.11b/g wireless link.

than tripled (*i.e.,* 300%) at PHY rates between 8 and 18 Mbps, and more than an order of magnitude (*i.e.,* 1000%) at PHY rates above 18 Mbps. Furthermore, the fluctuations of both TCP and CTCP performance under different link layer coding rates and modulation schemes (indicated by the changes in the 802.11 transmission rate) suggests that CTCP is much more robust to network and link layer changes than TCP, although more testing is required to verify this claim.

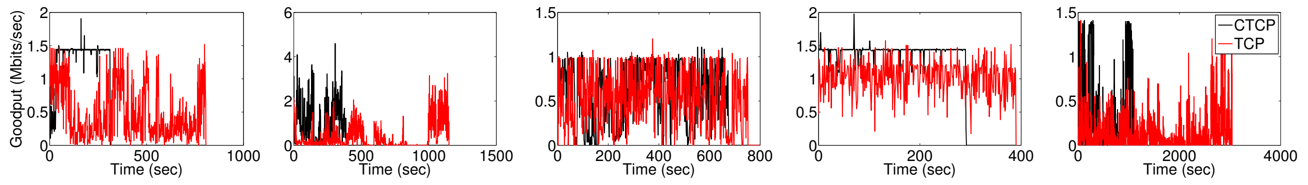
B. Hidden Terminal Interference

We now consider an 802.11 wireless link (configured similarly to that in Section VI-A) which is subject to hidden terminal interference. The hidden terminal is created by adding a third station to the network used in the last section. Carrier sense on the new terminal's wireless interface card is disabled and 1445 B UDP packets are transmitted with exponentially distributed inter-arrival times. The 802.11 transmit rates for both the hidden terminal and AP were set to 11 Mbps. Figure 8 plots the measured throughput on the downlink from the AP to a wireless client versus the mean transmit rate of the hidden terminal traffic. CTCP consistently obtains approximately twice (*i.e.,* 200%) the throughput of standard TCP (Cubic TCP) across a wide range of interference conditions.

C. Public WiFi Measurements

Finally, the performance of CTCP in a completely uncontrolled environment was measured to determine the effectiveness of our proposed congestion control. Measurements were collected at public WiFi networks in the greater Boston area by downloading a 50 MB file from a server (running Ubuntu 10.04.3 LTS) on MIT campus to a laptop (running Ubuntu 12.04.1 LTS) under the public WiFi hotspot. The default operating system settings are used for all of the network parameters. Figure 9 shows representative traces for five examples of these experiments. It is important to point out that standard TCP stalled and had to be restarted twice before successfully completing in the test shown in Figure 9c. CTCP, on the other hand, never stalled nor required a restart.

Each trace represents a different WiFi network that was chosen because of the location, accessibility, and perceived congestion. For example, the experiments were run over WiFi networks in shopping center food courts, coffee shops, and hotel lobbies. In Figures 9a - 9d, the WiFi network spanned a



(a) CTCP Time = 313 s, TCP Time = 807 s, Mean PLR = 4.28%, Mean RTT = 54.21 ms
(b) CTCP Time = 388 s, TCP Time = 1151 s, Mean PLR = 5.25%, Mean RTT = 73.51 ms
(c) CTCP Time = 676 s, TCP Time = 753 s, Mean PLR = 4.65%, Mean RTT = 106.31 ms
(d) CTCP Time = 292 s, TCP Time = 391 s, Mean PLR = 4.56%, Mean RTT = 50.39 ms
(e) CTCP Time = 1093 s, TCP Time = 3042 s, Mean PLR = 2.16%, Mean RTT = 208.94 ms

Fig. 9: Public WiFi Network Test Traces (CTCP in black, TCP in red). The download completion times, the mean packet loss rate (PLR), and mean RTT for each experiment are also provided.

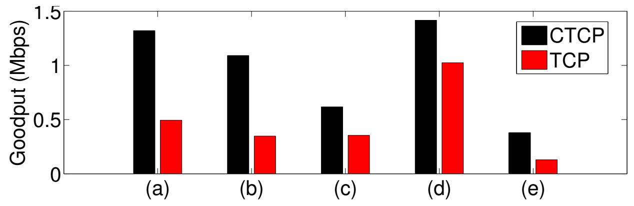


Fig. 10: Mean goodput for each of the experiments shown in Figure 9.

large user area increasing the possibility of hidden terminals; a scan of most of the networks showed > 40 active WiFi radios. The only experiment that had a small number of terminals (*i.e.,* five active radios) is shown in Figure 9e. The mean packet loss rate measured over all experiments was approximately 4%.

In each of the experiments, CTCP consistently achieved a larger average goodput and faster completion time. The average throughput for both CTCP and TCP is shown in Figure 10. Taking the mean throughput over all of the conducted experiments, CTCP achieves a goodput of approximately 750 kbps while standard TCP achieves approximately 300 kbps; resulting in a gain of approximately 2.5 (*i.e.,* 250%).

We emphasize the observed loss rates of approximately 4% in Figure 9 is quite high and unexpected, resulting in CTCP's significant performance gain over TCP. We believe that the loss rate is not only due to randomness but also due to congestion, interference, and hidden terminals. This is an area that would be worthwhile to investigate further. If our intuition is indeed correct, we believe that CTCP can greatly help increase efficiency in challenged network environments.

VII. Conclusions

We considered a new congestion control for transport layer packet streams that use error-correction coding to recover from packet losses with the following goals: increase throughput under high packet loss rates, operate under a wide range of network conditions, provide fairness to non-coded flows, and increase upper layer quality of service. We introduced a modified AIMD approach, developed an approximate mathematic model suited to performance analysis, and presented extensive experimental measurements both in the lab and in the "wild" to evaluate performance. In controlled lab experiments, we consistently observed reductions of more than an order of magnitude (*i.e.,* > 1000%) in completion times for both HTTP and streaming video flows when the link packet loss rate exceeds 5%. Finally, measurements using an 802.11 testbed and public WiFi hotspots, which highlights CTCP's performance under real-world scenarios, showed reductions in connection completion times of 100-300% compared with uncoded TCP. In terms of the potential improvements in the quality of service experienced by someone streaming video under a public WiFi hotspot with an average packet loss rate of 4%, CTCP would eliminate all buffer under-runs while approximately 50 buffer under-runs would occur using standard TCP (making the video almost unwatchable). We have shown that the application of the proposed congestion control to coded transport layer streams not only significantly increases throughput over a wide range of packet loss rates and RTTs, but also has positive implications for upper layer performance that directly affects the user's quality of service.

References

[1] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proceedings of IEEE*, vol. 99, pp. 490–512, March 2011.

[2] M. Kim, M. Médard, and J. Barros, "Modeling network coded TCP throughput: A simple model and its validation," in *Proceedings of ICST/ACM Valuetools*, May 2011.

[3] "IRTF ICCRG and NWCRG Meeting, Berlin, July 2014." http://www.ietf.org/proceedings/87/agenda/agenda-87-iccrg.

[4] J. Cloud, F. du Pin Calmon, W. Zeng, G. Pau, L. Zeger, and M. Médard, "Multi-path TCP with network coding for mobile devices in heterogeneous networks," in *Proceedings of VTC-Fall*, 2013.

[5] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle, "Loss-Tolerant TCP (LT-TCP): Implementation and experimental evaluation," in *MILCOM*, pp. 1–6, Oct 2012.

[6] "Wireless technologies for network service providers 2012-2013." Technicolor White Paper.

[7] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Network coded TCP (CTCP)," in ArXiv *http://arxiv.org/abs/1212.2291*.

[8] M. Kim, A. ParandehGheibin, L. Urbina, and M. Médard, "CTCP: Coded TCP using multiple paths," in ArXiv *http://arxiv.org/abs/1212.1929*, 2012.

[9] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, pp. 4413–4430, October 2006.

[10] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 133–145, 2000.

[11] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proceedings of INFOCOM*, 2006.

[12] R. N. Shorten and D. J. Leith, "On queue provisioning, network efficiency and the transmission control protocol," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 866–877, 2007.

[13] R. Shorten, F. Wirth, and D. Leith, "A positive systems model of TCP-like congestion control: asymptotic results," *IEEE/ACM Trans. Netw.*, vol. 14, pp. 616–629, 2006.