Congestion Control for Coded Transport Layers

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

***Abstract*—We consider congestion control for transport layer packet streams which use error-correction coding to recover from packet loss. We introduce a modified AIMD approach, develop an approximate mathematic model suited to performance analysis, and present extensive experimental measurements both in the lab and in the "wild" to evaluate performance. Our measurements highlight the potential for remarkable performance gains when using coded transports.**

I. Introduction

In this paper we consider congestion control for transport layer packet streams which 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]–[4]. Much of this has been driven by the ubiquity of wireless connectivity at the network edge. With the increasing density of wireless last hop deployments, interference is becoming a major contributor to packet erasures in unlicensed and white-space bands and can be the primary factor affecting overall network performance. In addition, there is a growing trend towards offload of cellular traffic onto 802.11 wireless links, leading to the use by outdoor smartphone clients of access points located in indoor hotspots (cafes *etc*) and vice versa, of outdoor municipal WiFi by clients located indoors (in shops *etc*). 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 as, unlike link layer changes, it ensures backward compatibility with legacy equipment. A recent industry study [5] estimates 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 technology is largely impractical due to the costs involved.

A key issue is, of course, congestion control. Congestion control is responsible for protecting the network from congestion collapse, for ensuring reasonably efficient use of available capacity, and for allocating this capacity in a roughly fair manner. When links may be lossy, packet loss is no longer synonmous with network congestion. When error-correction coding is used to recover from packet loss, packet retransmission is unnecessary and the trade-off between throughput and delay changes. When paths may have different levels of packet loss, notions of fairness need to be revisited (*e.g.* should a lossy path be allocated more or less throughput than a loss-free path sharing the same bottleneck link ?).

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).

In this paper we introduce a modified AIMD approach to congestion control for coded packet flows, develop an approximate mathematic model suited to performance analysis, and present extensive experimental measurements both in the lab and in the "wild" to evaluate performance. Our measurements highlight the potential for remarkable performance gains when using coded transports. In controlled lab experiments 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%. In measurements in the "wild" at public WiFi hotspots, we find 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 penalising standard TCP flows sharing the same links.

II. Preliminaries

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.* whether coded or uncoded). 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. A token allows the coded sender to transmit a packet (coded or uncoded). When the sender transmits a packet, the token is used.

Our focus here is on congestion control, but in order to carry out experimental performance evaluation we need to also specify the coding scheme. In this paper we consider the following coding scheme. Information packets are queued at the sender and transmitted in order. Using an estimate p of the path packet loss rate (based on ACK feedback from the receiver), after each block of $N = 32$ packets on average $N_c = N(\frac{1}{1-p} - 1)$ coded packets are transmitted. Typically N_c is not an integer, in which case $\lfloor N_c \rfloor$ coded packets are transmitted and an additional coded packet transmitted with probability $N_c - \lfloor N_c \rfloor$. Coded packets are constructed using an equiprobable random linear code in field $GF(2^8)$. In case of a decoding failure at the receiver, additional coded packets are transmitted for that block.

III. Congestion Control

Traditional TCP's AIMD congestion control increases the TCP sender's congestion window size $cwnd$ by α packets per RTT and multiplicatively decreases $cwnd$ by a backoff factor β on detecting packet losses within one RTT inducing a single $cwnd$ backoff. The usual values are $\alpha = 1$ when appropriate byte counting is used, and $\beta = 0.5$. On lossy links, repeated backoffs in response to noise losses rather than queue overflow can prevent $cwnd$ from increasing to fill the available link capacity. The behavior is well known and is captured, for example, in [6] in which $cwnd$ scales as $\sqrt{1.5/p}$, where p is the packet loss rate.

The basic issue here is that on lossy links, 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 signalling, for example via ECN, but this requires both network-wide changes and disabling of $cwnd$ backoff on packet loss. 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 [7] 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 (which is typically estimated as the lowest per packet RTT observed during the lifetime of a connection) and RTT is the current round-trip time.

This is similar to the approach considered in [8], which uses $\beta = RTT_{min}/RTT_{max}$ with the aim of making TCP throughput performance less sensitive to the level of queue provisioning. Indeed on links with only queue overflow losses, (1) reduces to the approach in [8] since $RTT = RTT_{max}$ (the link queue is full) when loss occurs. In this case, when 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, when queue overflow occurs the sum of the flows' throughputs must equal the link capacity B, $\sum_{i=1}^{n} tokens_i/RTT_i = B$ where n is the number of flows. After backoff according to (1), the sum-throughput becomes $\sum_{i=1}^{n} \beta_i tokens_i/RTT_{min,i} = B$ when the queue empties. That is, the choice (1) for β decreases the flow's $tokens$ so that the link queue just empties and full throughput is maintained.

On lossy links (with losses in addition to queue overflow losses), use of RTT in (1) adapts β. When a network path is under-utilized, $RTT = RTT_{min}$ (therefore, $\beta = 1$ and $\beta \times tokens = tokens$) and $tokens$ is not decreased on packet loss. Hence, $tokens$ is able to grow, despite the presence of packet loss. Once the link starts to experience queueing delay, $RTT > RTT_{min}$ and $\beta < 1$, 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 decreasing $tokens$, the sum-throughput is at least (when all flows backoff their $tokens$) $\sum_{i=1}^{n} \beta_i tokens_i/RTT_{min,i} = B$ when the queue empties. That is, (1) adapts β to maintain full throughput.

Although we focus on using (1) in combination with linear additive increase (where α is constant), we note that this adaptive backoff approach can also be combined with other types of additive increase including, in particular, those used in Cubic TCP and Compound TCP.

IV. Throughput Performance Modeling

Consider a link shared by n flows. Let B denote the link capacity and T_i the round-trip propagation delay of flow i. We will assume that the queueing delay is negligible, i.e. the queues are small or the link is sufficiently lossy that the queue does not greatly fill. We also assume that any differences in the times when flows detect packet loss (due to the differences in path propagation delay) 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 $tokens$. Let $w_i(k)$ denote the $tokens$ of flow i immediately before the k-th network backoff event and $s_i(k) = w_i(k)/T_i$ the corresponding throughput. We 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 (1) otherwise. The time $T(k)$ is also a random variable, with distribution determined by 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 the δ_j are i.i.d with erasure probability p. Then $Prob(T(k) \le t) = 1 - (1-p)^{N_t(k)}$ 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 $[T(k), T(k) + T_i]$, which 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 $s_i(k)$, $i = 1, \cdots, n$, analysis of the network dynamics is generally challenging. When the backoff factor $\tilde{\beta}_i(k)$ is stochastically independent of the flow rate $s_i(k)$, the analysis is then relatively straightforward. 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) [9]. 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 (and so to RTT in (1)) is small. Further, as we will see later, experimental measurements indicate that analysis using the assumption of independence accurately predicts performance over a range of other network conditions, and so results are insensitive to this assumption.

Given independence, from (2),

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

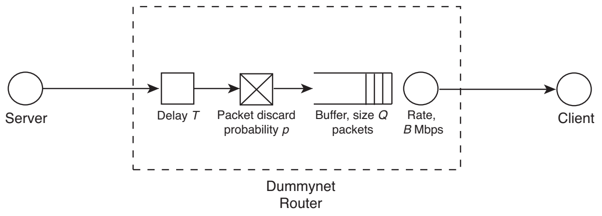


Fig. 1: Schematic of experimental testbed.

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 (3) we have

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

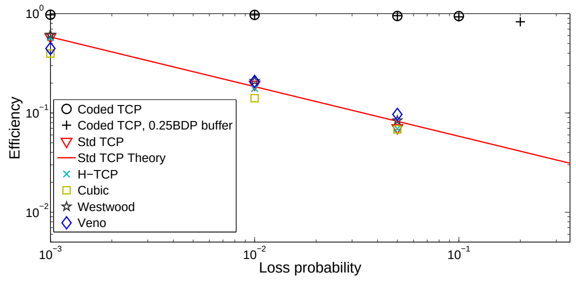
Since the factor $\mathbb{E}[T]$ is common to all flows, the fraction of link capacity obtained by flow i is determined by $\frac{\tilde{\alpha}_i}{1-\mathbb{E}[\tilde{\beta}_i]}$.

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

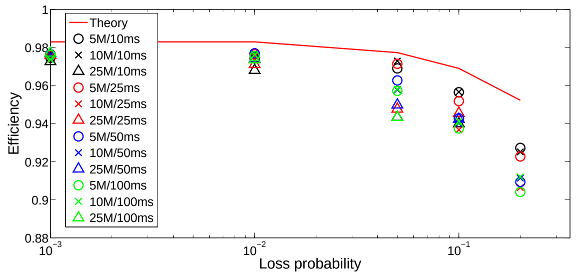
Fairness between flows with different RTTs: When flows i, j have different round trip times $T_i \neq T_j$ but the same mean backoff factor, the ratio of their throughputs is $\frac{\mathbb{E}[s_i]}{\mathbb{E}[s_j]} = (\frac{T_j}{T_i})^2$. Observe that this is identical to standard TCP behavior [9].

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. Hence, 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), this will affect fairness through $\mathbb{E}[\tilde{\beta}_i]$.

Friendliness: The model (2) is sufficiently general enough to include AIMD with fixed backoff factor, as used by standard TCP. We consider two cases. First, when the link is loss-free (the only losses are due to queue overflow) and all flows backoff when the queue fills, then $1 - \mathbb{E}[\tilde{\beta}_i] = 1 - \beta_i(k)$. For a flow i with fixed backoff of 0.5 and a 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 (4) when the flows have the same RTT. When $\beta_j = T_j/RTT_j = 0.5$, the throughputs are equal. Since $RTT_j = T_j + q_{max}/B$ where q_{max} is the link buffer size and B the link rate, $\beta_j = 0.5$ when $q_{max} = BT_j$ (i.e., the buffer is size at the bandwidth-delay product). The second case we consider here is when the link has i.i.d packet losses with probability p. When p is sufficiently large so that the queue rarely fills, queue overflow losses are rare and the throughput of a flow i with fixed backoff of 0.5 is accurately modeled by the Padhye model [6]. That is, the throughput is largely decoupled from the behavior of other flows sharing the link (since coupling takes place via queue overflow) and, in particular, this means that flows using adaptive backoff do not penalize flows which use fixed backoff. We present experimental measurements confirming this behavior in Section V-C.



(a) Link 25Mbps, RTT 20ms



(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 Equation (5).

V. Experimental Measurements

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 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, and so 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 300s. Data traffic is generated using `rsync` (version 3.0.4), HTTP traffic using `apache2` (version 2.2.8) and `wget` (version 1.10.2), video traffic using `vlc` as 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.

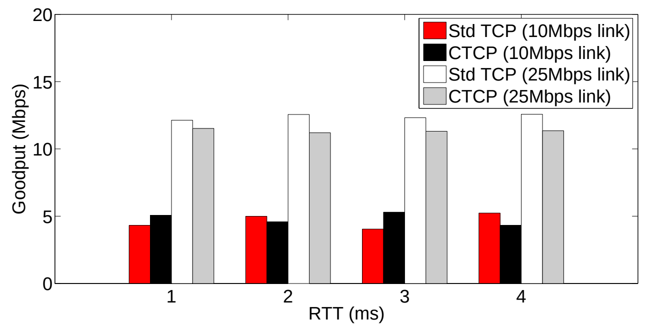
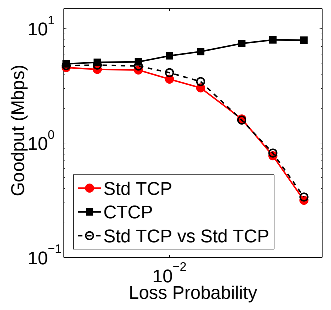
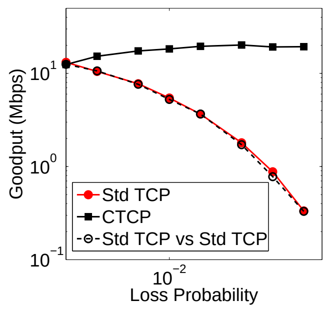


Fig. 3: Goodput for a standard TCP and a CTCP flow sharing a loss-free link; results are shown for 10Mbps and 25Mbps links with varying RTTs.



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



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

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).

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 25Mbps link with 25ms RTT and a bandwidth-delay product of buffering. Baseline data is shown for standard 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 predicted by the popular Padhye model [6]. It can be seen that the measurements for standard TCP are in good agreement with the Padhye model, as expected. Also that Cubic TCP, H-TCP, TCP Westwood, and TCP Veno closely follow the standard TCP behavior. Again, this is as expected since the link bandwidth-delay product of 52 packets lies in the regime where these TCP variants seek to ensure backward compatibility with standard TCP. Observe that the achieved goodput decreases rapidly with increasing loss rate, falling to 20% of the link capacity when the packet loss rate is 1%. This feature of standard TCP is, of course, well known. Compare this, however, with the efficiency measurements for CTCP, which are shown in Figure 2a and also given in more detail in Figure 2b. The goodput is $>96\%$ of link capacity for a loss rate of 1%, a roughly five-fold increase in goodput compared to standard TCP.

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

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

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. This value η is the mean number of such forward-transmitted coded packets that are unnecessary (because there are fewer then n erasures).

The efficiency achieved by CTCP is also insensitive to the buffer provisioning, as discussed in Section III. 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, it can be seen that 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 the CTCP flow and the standard TCP flow 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).

For comparison, in Figure 4, we also show (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 by a standard TCP flow (dotted line) when competing against another standard TCP flow is close to that achieved when sharing the link with a CTCP flow (solid line). This demonstrates that CTCP does not penalize the standard TCP flow.

D. *Application Performance*

1) Web: Figure 5 shows measurements of HTTP request completion time against file size for standard TCP and CTCP. The HTTP requests are generated using `wget` and the response is by an `apache2` web server. Note the log scale on the y-axis.

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

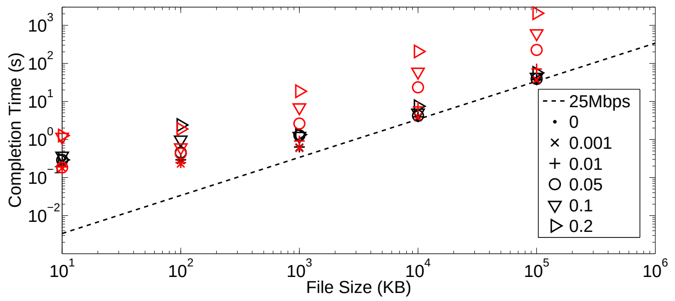
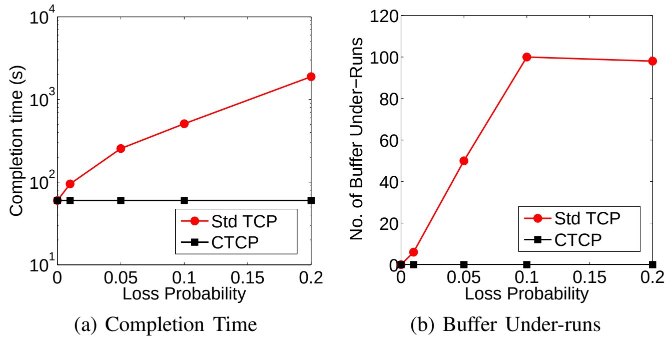


Fig. 5: Measured HTTP request mean completion time against file size over 25Mbps link with RTT = 10ms. Data is shown for standard TCP (red) and CTCP (black) for a range of loss rates. Error bars are comparable in size to the symbols used in the plot and so are omitted.



(a) Completion Time (b) Buffer Under-runs

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

loss rate. For a 1MB connection, the completion time with standard TCP increases from 0.9s to 18.5s as the loss rate increases from 1% to 20%, while for a 10MB connection the corresponding increase is from 7.1s to 205s. Observe that the completion time is reduced by more than 20× (2000%) for a 1MB connection and by almost 30× (3000%) for a 10MB connection.

2) Streaming Video: Figure 6 plots performance measurements for streaming video for a range of packet loss rates on a 25Mbps link with RTT equal to 10 ms. A `vlc` server and client are used to stream a 60s 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 60s and is insensitive to the packet loss rate. In contrast, the playout time with standard TCP increases from 60s to 95s when the loss rate is increased from 0% to 1%, and increases further to 1886s (31 minutes) as the loss rate is increased to 20% (more than 30× slower than when using CTCP). Figure 6b plots measurements of playout buffer under-run events at the video client. It can be seen that there are no buffer under-run events 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

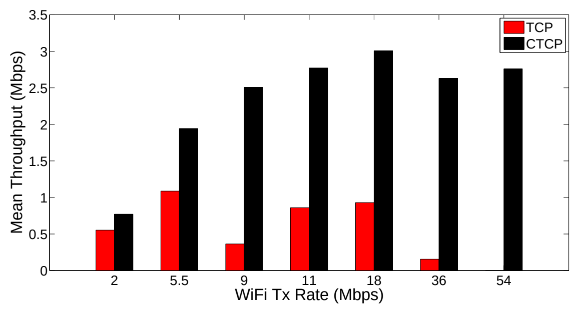


Fig. 7: Measurements of mean throughput vs wireless PHY rate used with standard TCP (Cubic TCP) and CTCP on an 802.11 link affected by microwave oven interference.

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%.

VI. Real-world Performance

A. Microwave Oven Interference

We begin by considering an 802.11b/g wireless client downloading from an access point over a link subject to interference from a domestic microwave oven (MWO). The wireless client and AP were equipped with Atheros 802.11 b/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 microwave oven used was a 700 W Tesco MM08 17L. Unless otherwise stated, the default operating system settings are used for all network parameters. The wireless client used rsync to download a 50MB file via the AP. Figure **??** (shown in the introduction) presents spectrum analyzer (Rohde & Schwarz FSL-6) measurements illustrating both the MWO interference and packet transmissions on the wireless link. The MWO operates in the 2.4 GHz ISM band, with significant overlap ($> 50\%$) with the WiFi 20 MHz channels 6 to 13. The MWO interference is approximately periodic, with period 20ms (i.e. 50Hz) and mean pulse width 9ms (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 vs the PHY rate used on the downlink. Data is shown using standard TCP (in this case Cubic TCP, the Linux default variant) and CTCP. Data is not shown for a PHY rate of 1Mbps as the packet loss rate was close to 100% – this is because at this PHY rate, the time to transmit a 1500B frame is greater than the interval between MWO interference bursts and so almost all frames are damaged by the interference. It can be seen that the throughput achieved by standard TCP rises slightly as the PHY rate is increased from 1Mbps to 5.5Mbps, but then decreases to zero for PHY rates above 36Mbps (due to channel path losses). In comparison, when using CTCP the throughput achieved is approximately doubled (200%) at a PHY rate of 5.5Mbps, more than tripled (300%) at PHY rates of 8, 11 and 18 Mbps and increased by more than an order of magnitude (1000%) at a PHY rates above this. Furthermore, the fluctuations of both

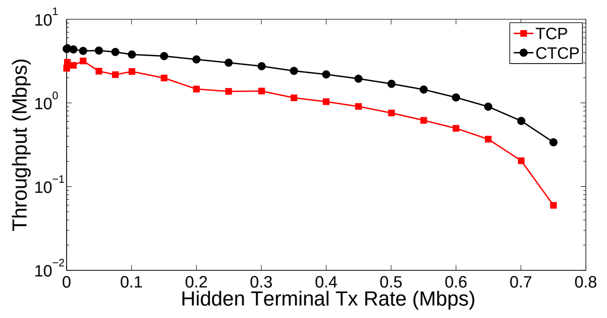


Fig. 8: Throughput vs intensity of hidden terminal interference when using standard TCP (Cubic TCP) and CTCP over an 802.11b/g wireless link.

TCP and CTCP performance under different link layer coding rates and modulation schemes (indicated by the changes in the WiFi transmission rate) suggests that CTCP's performance is much more robust to network changes than that of TCP's performance, 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 created in the last section. Carrier sense on the terminal's wireless interface card is disabled and 1445B 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. Unless otherwise stated, the default operating system settings are used for all of the remaining network parameters. 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. It can be seen that CTCP consistently obtains approximately twice (200%) the throughput of standard TCP (Cubic TCP) across a wide range of interference conditions.

C. Public WiFi Measurements

Measurements were collected at various public WiFi networks in the greater Boston area by downloading a 50 MB file from a server (running Ubuntu 10.04.3 LTS) located on the 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 network parameters on client and server. 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 large user area increasing the possibility of hidden terminals; a

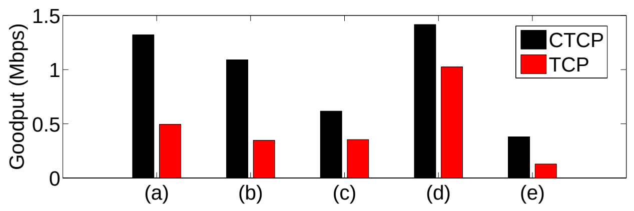


Fig. 10: Mean goodput for each of the experiments shown in Figure 9.

scan of most of the networks showed > 40 active WiFi radios, which also increases the probability of collision losses. 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%.

It can be seen that 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 (250%).

We emphasize the observed loss rates of approximately 4% in Figure 9, which is quite high and unexpected; resulting in CTCP's significant performance gain over TCP's. 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 consider congestion control for transport layer packet streams which use error-correction coding to recover from packet loss. We introduce a modified AIMD approach, develop an approximate mathematic model suited to performance analysis, and present extensive experimental measurements both in the lab and in the "wild" to evaluate performance. In controlled lab experiments, we consistently observe 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%. Measurements in public WiFi hotspots demonstrate reductions in connection completion times of 100-300% compared with uncoded TCP.

References

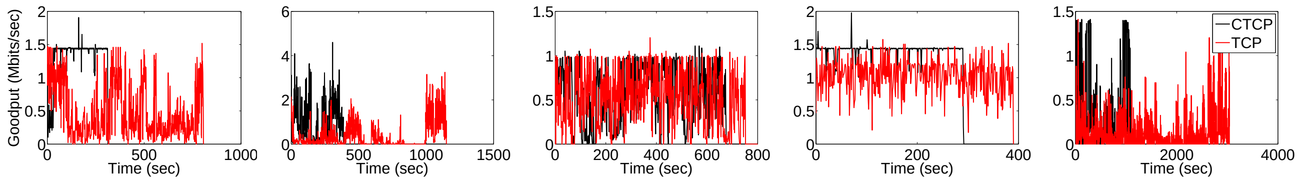
[1] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proceedings of IEEE*, vol. 99, pp. 490–512, March 2011.

[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 Meeting, Berlin, July 2014." http://www.ietf.org/proceedings/87/agenda/agenda-87-iccrg.

[4] "IRTF NWCRG Meeting, Berlin, July 2014." http://www.ietf.org/proceedings/87/agenda/agenda-87-nwcrg.

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



(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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.