A TCP-Friendly Rate Adjustment Protocol for Continuous Media Flows over Best Effort Networks CMPSCI Technical Report TR 98-047 [*]

Jitendra Padhye[†], Jim Kurose, Don Towsley,
University of Massachusetts at Amherst,
Amherst, MA 01002
{jitu, kurose, towsley}@cs.umass.edu

Rajeev Koodli
Nokia Research Center,
Burlington, MA 01803
rajeev.koodli@research.nokia.com

October 23, 1998

Abstract

As networked multimedia applications become widespread, it becomes increasingly important to ensure that these applications can coexist with current TCP-based applications. The TCP protocol is designed to reduce its sending rate when congestion is detected. Networked multimedia applications should exhibit similar behavior, if they wish to co-exist with TCP-based applications [7]. Using TCP for multimedia applications is not practical, since the protocol combines error control and congestion control, in order to ensure reliability. In this paper we present a protocol that operates by measuring loss rates and round trip times and then sets the transmission rate to that which TCP would achieve under similar conditions. The analysis in [12] is used to determine this ”TCP-friendly” rate. We evaluate the protocol under various traffic conditions, using simulations and implementation.

1 Introduction

As networked continuous media (CM) applications and stored WWW-based CM servers become more widespread, it becomes increasingly important to ensure that they be able to co-exist with each other and with current TCP-based applications. A key component of such a co-existence is the incorporation of some form of congestion control that results in a reduction of transmission rate in the face of network congestion. Many current CM applications simply transmit CM data at the rate at which it was encoded, regardless of the congestion state of the network.

[*]This material is based upon work supported by the National Science Foundation under Grant Nos. CDA-9502639 and NCR-9508274. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[†]Currently working for Nokia Research Center, Burlington, MA 01803

Several considerations come into play when designing a CM congestion control protocol. First, because CM applications are both loss-tolerant and time-sensitive, the transmission rate might best be adapted in a manner that is cognizant of the loss resilience and timing constraints of the application [13, 17]. Second, since the applications must co-exist with TCP-based applications, CM congestion control algorithms should adapt their rate in a way that "fairly" shares congested bandwidth with such TCP applications. One definition of "fair" is that of TCP "friendliness" [7] – if a CM connection shares a bottleneck link with TCP connections, traveling over the same network path, then the CM connection should receive the same share of bandwidth (i.e., achieve the same throughput) as a TCP connection. A first step then in developing a CM congestion control protocol, is to investigate congestion control protocols that set their transmission rate in a TCP-friendly manner. Once such a TCP-friendly protocol is developed, it can then be modified to account for the timeliness requirements of CM data delivery, perhaps with some loss of "friendliness."

Three possible approaches can be identified for achieving TCP friendliness. First, TCP itself could be used. One drawback of this approach is that TCP provides not only congestion control but also error control and ordered delivery to the application. As noted above, CM applications are loss resilient and time-sensitive – they may not require recovery of a lost packet or may choose to recover in a delay-sensitive manner [17, 13]. A second approach is to not perform error recovery, but to still *exactly* mimic the sending behavior of TCP, (that is, send the next packet in the CM flow whenever TCP would send either an original or a retransmitted packet). One might term this a "TCP-exact" approach. A potential drawback of this approach is that it would be difficult to determine the consequences of deviating from TCP-exactness when accommodating the timeliness requirements of CM delivery. Altering the TCP-exact sending of any packet will have unknown effects on the delay and reception/loss of that packet, its acknowledgment, and future packets as well. [5] takes a window-based TCP-exact approach towards CM congestion control.

The third approach is to develop a congestion control algorithm that controls the sending rate in a manner that is roughly equivalent to TCP. Specifically, if a TCP connection achieves throughput X under given network conditions and measured over a given interval length, then the CM protocol should also have throughput X over an interval of the same length and under the same network conditions. As noted above, the actual transmission rate, X, can be determined by using a model-based characterization of TCP throughput in terms of network conditions such as round trip times and loss rates. In this paper, we take this third approach, using the formula for TCP throughput derived in [12]. The formula in [12] requires the mean round trip time and the loss rate to be known; these values will be measured by the TCP-friendly protocol we develop.

We believe there are several advantages of taking a formula-based approach towards CM congestion control. First, by changing the formula, one can easily adjust the performance of the formula compared to TCP. In the future, TCP and CM flows may be treated separately in the network, perhaps using a scheme such as [2]. The formula-based approach can also be easily modified for an environment in which CM flows compete only against one another, and not against TCP. In [23], it has been shown that such an approach is more suitable for multicasting. Thus, a formula-based approach based on an abstract TCP characterization can be viewed as a first step towards developing a comprehensive solution to the problem of congestion control for CM flows.

In this paper, we describe a formula-based, TCP-friendly congestion control protocol known as CMTCP. The protocol operates by measuring the loss rate and round-trip times of the CM flow, and then setting its transmission rate to be that which TCP would achieve under similar circumstances. We are interested in the general problem of congestion control for CM flows, which include aspects of buffer management at the sender and the receiver, co-existence with other CM flows in the network and co-existence with TCP traffic. In this paper we focus our attention on achieving TCP-friendliness, while retaining enough flexibility to develop a more general mechanism for CM congestion control – one that accommodates timing constraints and loss resilience.

We evaluate the CMTCP protocol using simulation as well as implementation, experimentation and measurement in the Internet. The simulations are used study the behavior of the protocol under controlled conditions. We find the protocol is "fair" to TCP (in a sense we make precise shortly) and to other sessions running CMTCP. The implementation and experimentation involve over 300 experiments that we carried out at various times between several hosts located in the US and UK. Our experimentation confirms the simulation results and show that the formula-based approach we examine is indeed practical.

The rest of this paper is organized as follows. In Section 2, we present an overview of related work reported in the literature, followed by a description of our protocol and its advantages. In Section 3, we present simulation studies of our protocol. In Section 4, we present results from a "real-world" implementation of the protocol. In Section 5 we discuss some of our design choices. Section 6 concludes the paper.

2 Rate Adjustment Protocols

Several TCP-friendly rate adjustment protocols have been recently reported in the literature [23, 24, 16, 19]. Of these, [23] and [24] are specific to multicast applications, while [16] and [19] are more unicast oriented. We now briefly review each of these four schemes, describe the new CMTCP protocol, and show how it overcomes some of the shortcomings in earlier work.

2.1 Previous Work

The TCP-friendly protocols reported in [23, 24, 16, 19] are based (either explicitly or implicitly) on the TCP characterization first reported in [7] and later formalized in [8, 11]. The characterization states that in absence of timeouts, the steady state throughput of a long-lived TCP connection is given by:

$$\text{Throughput} = \frac{C}{R * \sqrt{p}} \tag{1}$$

where C is a constant that is usually set to either 1.22 or 1.31, depending on the receiver[1], R is the round trip time experienced by the connection, and p is the expected number of window reduction events per packet sent. Note that the throughput is measured in terms of packets/unit time. Also note that p is *not* the packet loss rate, but is the frequency of loss indications per packet sent [8]. The packet loss rate provides an *upper bound* on the value of p, and may be used as an approximation. The key assumption

[1]Specifically, depending on whether or not receiver uses delayed acknowledgments.

behind the characterization in (1) is that timeouts [20] do not occur at all. Consequently, it is reported in [8] that (1) is not accurate for loss rates higher than 5%. As the formula does not account for timeouts, it typically *overestimates* the throughput of a connection as loss rate increases. Data presented in [8, 12] shows that timeouts account for a large percentage of window reduction events in real TCP connections, and that they affect performance significantly.

In [23] the authors propose a multicast congestion control scheme in which the data is transmitted in a "layered" manner over different multicast groups. The more layers a receiver joins, the more data it receives, and the higher the "utility" of the received data to the CM application. In [23] the receivers compute round trip times and estimate the packet loss rate p, and use (1) to compute the "TCP-friendly" rate at which they should receive the data. Based on this estimate, and the knowledge of the layering schemes, each receiver can dynamically decide to join or leave certain multicast groups to adjust the rate at which it receives the data. In [24], the authors propose a similar scheme in which the layers have data rates that are fixed multiples of a base rate, and a TCP-like effect (additive increase, multiplicative decrease) is achieved by using strict time limits on when a receiver might join or leave a group. The analysis of the algorithm yields a throughput characterization that is similar to (1). Both schemes rely on data layering and the assumption that an incremental increase in the data rate results in an incremental increase in utility; these two properties do not necessarily hold for all types of CM encodings. In addition, determining round trip times in a multicast setting is a difficult task, as noted in [23].

In [19] the authors propose a scheme that is suitable mainly for unicast applications, but may be modified for multicast application. The scheme relies on regular RTP/RTCP reports [18] sent between the sender and the receiver to estimate loss rate and round trip times. In addition, they propose modifications to RTP that allow the protocol to estimate the bottleneck link bandwidth using the packet-pair technique proposed in [1]. An additive increase/multiplicative decrease scheme based on these three estimates (loss rate, round trip delay, and bottleneck bandwidth) is then used to control the sending rate. The scheme has several tunable parameters whose values must be set by the user. In addition, the scheme is not "provably" TCP-friendly, although TCP-friendliness is evidenced in the few simulations reported in the paper. In [16] the authors propose an additive increase/multiplicative decrease rate control protocol that uses ACKs (in a manner similar to TCP) to estimate round trip times and detect lost packets. The rate adjustment is done every round trip time. The authors also propose to use the ratio of long-term and short-term averages of round trip times to further fine tune the sending rate on a per-packet basis. The protocols reported in [19] and [16] do not explicitly use (1) to control their rates. The work in [7, 8, 11], however, has shown that the effect will be similar. As mentioned earlier, results from the expression in (1) is generally not valid for loss rates higher than 5%, and any rate control protocol based on this equation will not be "TCP-friendly" at loss rates higher than 5%. While [19] ignores this problem, in [16] the authors mention that their work is targeted towards a future scenario in which SACK TCP [3] and RED [4] switches will be widely deployed, reducing the probability of timeouts. However, in the present Internet, TCP-Reno [20] is the predominant protocol and very few RED switches have been deployed.

In the next section we propose a new protocol that achieves TCP friendliness in a more "real world" scenario that includes competing TCP-Reno connections, drop-tail switches and diverse background traffic conditions. We name this protocol CMTCP (Continuous Media TCP)[2].

2.2 The CMTCP Protocol

The CMTCP protocol is a rate-adjustment congestion control protocol that is based on the TCP characterization proposed in [12]. Unlike [7, 8, 11], the characterization in [12] takes into account the effects of timeouts, a consideration that is particularly important when TCP-Reno (one of the most widely deployed versions of TCP) is used with drop-tail routers. Drop-tail routers tend to produce correlated losses. If a TCP-Reno connection encounters correlated losses, it tends to experience a significant number of timeouts [3]. In [12] the authors quantify this phenomenon and its effects on throughput. The resulting analytic characterization of TCP throughput can stated as follows:

$$\text{Throughput} \approx f(W_{max}, R, p, B) \tag{2}$$

where throughput is measured in packets per unit time, W_{max} is the receiver's declared window size, R is the round trip time experienced by the connection, p is the loss rate (or, more accurately, the frequency of loss indications per packet sent) and B is the base timeout value [20]. A complete statement of the formula is presented in the Appendix.

The CMTCP protocol has two parts: a sender-side protocol and a receiver-side protocol. The sender-side protocol works in *rounds* of duration M time units. We call M the *recomputation interval*. At the beginning of each round, the sender computes a TCP-friendly rate (we will shortly describe this computation in detail), and sends packets at that rate. Each packet carries a sequence number and a timestamp indicating the time the packet was sent. The receiver acknowledges each packet, by sending an ACK that carries the sequence number and timestamp of the packet it is acknowledging. Consider an ACK for a packet whose sequence number is k. In addition to the sequence number and the timestamp, the ACK also carries a bit vector of 8 bits indicating whether or not each of the previous 8 packets $(k-7\ldots k)$ was received. The sender processes these ACKs to compute sending rate for the next round.

Let us now consider the sending rate computation in detail. Consider round i. Let r_i be the sending rate for this round, R_i be the estimate of the round trip time for this round and B_i be the estimate of the base timeout value for this round. The number of packets to be sent in this round is $n_i = r_i * M$. The n_i packets are clocked out uniformly during the round[3]. As noted earlier, packets carry a sequence number and a timestamp indicating the time the packet was sent. The sender keeps a log of all packets it has sent in this round. The log contains two entries for each packet. The first entry indicates whether the packet has been *(i)* received and has been acknowledged by the receiver; *(ii)* presumed lost; *(iii)* of unknown status (neither ACKed nor yet presumed lost). We call this the "received status" of the packet. The second entry consists of a value that is equal to the time the packet was sent plus the current base timeout value. We call this the "timeout limit" for the packet.

As the sender sends packets, it also receives ACKs from the receiver. Consider an ACK carrying sequence number k that is received by the sender at time t_k. Let the timestamp carried by the ACK be

[2]Or, Call Me TCP ....

[3]In simulation studies, it is possible clock out packets evenly over the entire duration of the round. This is not possible in actual implementation, due to limited accuracy of timers. We discuss this further in Section 4.

s_k. The sender updates the lost/received status of packets $(k - 7 \dots k)$ using the bit vector available in the ACK. The sender also updates the current round trip time estimate and base timeout using the difference $t_k - s_k$. This update is done exactly as in TCP; see [21] (Chapter 21), and [22] (Chapter 25) for the details of the computation. At the end of the i^{th} round, the sender computes r_{i+1}, R_{i+1} and B_{i+1} as follows:

- R_{i+1} and B_{i+1} are assigned the current estimates of round trip time and base timeout value respectively.
- The computation of the sending rate, r_{i+1}, proceeds as follows. Let the current time be t_i. Let j be the packet with the smallest sequence number, whose received status was "unknown" at the end of round $i - 1$, l be the last packet sent and a be the highest sequence number for which we have received an ACK. Then any packet whose sequence number lies between j and l, (both included) and whose timeout limit is less than t_i, is marked as lost. Also, any packet whose sequence number lies between j and a (both included), and whose received status is "unknown" is marked as lost. Let x_i be the number of packets marked as "received" between j and a, and let y_i be the number of packets marked as "lost" between j and a. Then:
  - If $y_i = 0$, then no packets were lost and:

    $$r_{i+1} = 2 * r_i$$

    Hence, when no packets are lost in a round, packets are sent twice as fast in the next round. We will discuss this feature more in Section 5.
  - Otherwise, $y_i \neq 0$. Let $p_i = \frac{x_i}{x_i + y_i}$. In this case, the rate for round $i + 1$ is

    $$r_{i+1} = f(W_{max}, R_{i+1}, p_i, B_{i+1})$$

    where f is defined in (2). It is here that the analytic characterization in [12] comes into play.
- If the computed value of r_{i+1} is such that:

  $$r_{i+1} < 1/R_{i+1}$$

  Then

  $$r_{i+1} = 1/R_{i+1}$$

The starting value, r_0 can be set to any reasonable value. We have found that for sufficiently long flows, and for reasonable values of M, the value of r_0 has little impact on the performance of the protocol. For all simulations and experiments described in this paper, we set this value to 40 packets/second. The values of R_0 and B_0 are set in a manner similar to TCP [21, 22].

CMTCP has no in-built error recovery mechanisms. This allows the CM applications to choose an error control strategy that is appropriate for the given media type. An important feature of any transmission control protocol is "self-limitation" [16]. This means that if the protocol starts experiencing 100% or near 100% losses, its sending rate should be reduced to almost zero. TCP achieves this via timeouts and eventual closedown of the connection. The CMTCP protocol currently reduces its sending rate to one packet per round trip time, in case of 100% loss. The key question is how frequently the sender

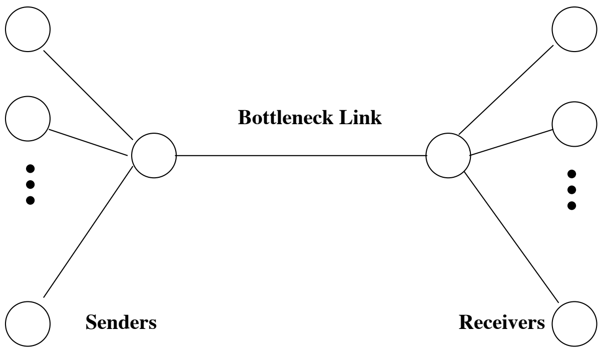


Figure 1: Simulation Topology

should re-compute the rate. In the current version of CMTCP, we recompute this rate at fixed intervals of time (i.e. fixed value of M). This value remains constant throughout the duration of the flow. We have explored other alternatives, some of which we discuss in Section 5.

3 Simulation Results

In this section we present simulation studies of the protocol proposed in the previous section. We have used the ns simulator [9] for our simulations. There are two main challenges for any simulation study of this nature: how to select appropriate network topologies and how to effectively model the background traffic. Several difficulties in this regard are pointed out in [15]. In our simulations, we use a simple topology to uncover and illuminate the important issues; our experiments with CMTCP over the Internet test its use in "real-world" scenarios. The simulated network topology assumes a single shared bottleneck link, as shown in Figure 1. The sources are arranged on one end of the link and the receivers on the other side. All links except the bottleneck link are sufficiently provisioned to ensure that any drops/delays that occur are only due to congestion at the bottleneck link. All links are drop-tail links. Many previous studies [3, 4, 19, 16] have used similar topologies.

The problem of accurately modeling background traffic is more difficult. We consider three principle kinds of traffic: infinite-duration FTP-like connections, medium-duration FTP -like connections and self-similar UDP traffic. The infinite-duration FTP connections allow us to study the steady-state behavior of our protocol. Medium-duration FTP connections introduce moderate fluctuations in the background traffic. Finally, self-similar UDP traffic is believed to be a good model for short TCP connections such as those resulting from web traffic [14, 25]. In the rest of this section, we describe each traffic scenario in detail and present the results from our simulation study.

3.1 Constant Bottleneck Bandwidth Share

In this scenario we consider traffic made up entirely of equal numbers of infinite-duration TCP connections and infinite-duration CM connections. These latter connections are controlled by CMTCP and always have data to send. All connections start at the beginning of simulation and last until the simulation ends. The aim here is to study CMTCP's steady state behavior. If CMTCP performs well (i.e., in a TCP-friendly manner), the TCP and CMTCP connections should see approximately the same throughput.

For these simulations, we vary the total number of flows in the network between 10 and 50. The bottleneck bandwidth is computed by multiplying the total number of flows by 4Kbps. The packet size for each flow is fixed at 100 bytes. The buffer size at the bottleneck link was set in each case to four times the bandwidth-delay product. These settings of packet and buffer sizes allow the TCP connections to have "reasonable" window sizes [16] and exhibit the full range of behavior such as slow start and congestion avoidance. Each experiment is repeated for various values of the recomputation interval, M. The initial sending rate, r_0, for all CMTCP connections was set to approximately 40 packets/second. When multiple TCP connections are simulated over a single bottleneck link, the connections can become synchronized, and the observed simulation results are biased. We take two measures to prevent such synchronization. First, we start the connections at slightly different times. Second, before each packet is sent out, a small random delay is added to simulate processing overhead. These measures are applied to both TCP and CMTCP connections.

To measure CMTCP performance, we must choose an appropriate fairness metric. Let the total number of monitored flows in a given simulation be k. Let k_c denote the total number of CM connections and k_t denote the total number of TCP connections ($k_c = k_t = k/2$). We denote the throughput of the k_c CMTCP connections by $T_1^c, T_2^c, \ldots T_{k_c}^c$ and that of the TCP connections by $T_1^t, T_2^t, \ldots T_{k_t}^t$ respectively. Define:

$$T_C = \frac{\sum_{i=1}^{k_c} T_i^c}{k_c}$$

$$T_T = \frac{\sum_{i=1}^{k_t} T_i^t}{k_t}$$

The performance metric of interest is the "friendliness ratio," F:

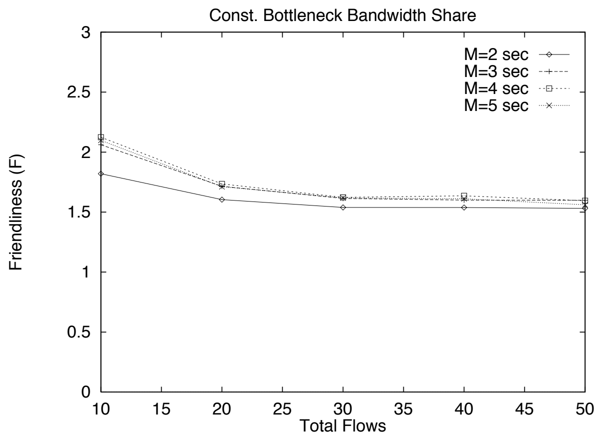
$$F = \frac{T_C}{T_T}$$

Another metric for measuring performance is the "equivalence ratio," E:

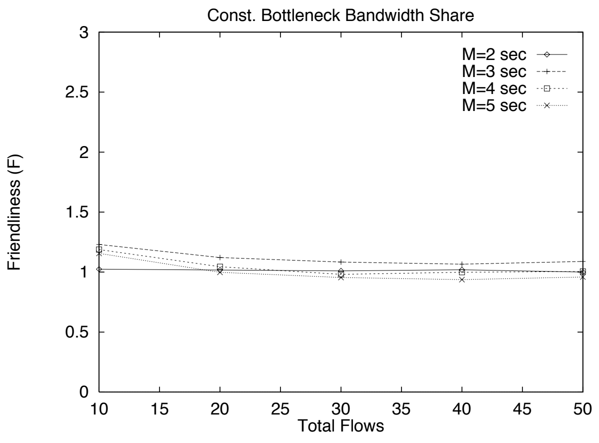
$$E = \max(\frac{T_T}{T_C}, \frac{T_C}{T_T})$$

Note that the value of E is always ≥ 1. E gives a better visual representation of the equivalence of the two protocols. However, this metric will distort any trend that might be present in the ratio of the two throughputs as we vary various parameters. For example, a decreasing value of F as a function of some system parameter will not always result in a decreasing value of E. Thus, we use F as the fairness metric whenever we re interested in $trends$, and use E otherwise.

In Figures 2(a) and 2(b), we present the results from our simulation for the cases when the bottleneck link propagation delay was 20ms and 50ms. as M is varied between 2 and 5 seconds. The length of each simulation was 1000 seconds, and the throughput of all connections was measured at the end of the simulation. Each data point is an average of three experiments. It can be seen that with steady state background traffic, the protocol is able to maintain a friendliness ratio close to 1, for bottleneck



(a) Bottleneck Delay 20ms



(b) Bottleneck Delay 50ms

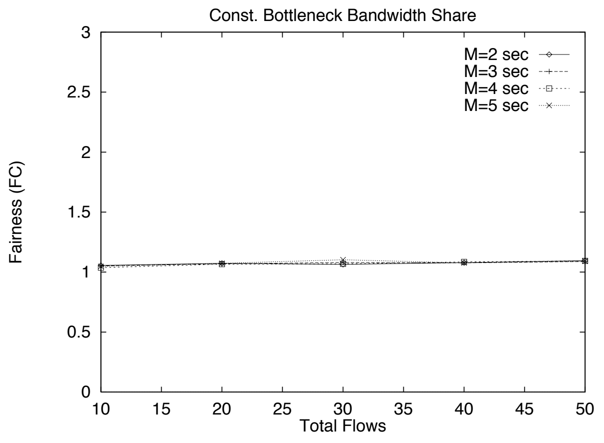
Figure 2: Friendliness (F), Constant Bottleneck Bandwidth Share

link propagation delays of 50ms. For small bottleneck propagation delays (i.e., 20ms) and hence small round trip times, we notice that, CMTCP behaves more aggressively than TCP. We conjecture that this is due to the fact that with lower round trip times, TCP reacts to losses and small changes in traffic fluctuations more quickly. On the other hand, the CMTCP protocol computes losses over a large number of packets sent, thus smoothing out the loss rate and achieving a higher throughput. For a given round trip time, however, we can see that the recomputation interval (M) does not have a significant effect on the friendliness ratio. This is due to the fact that the background traffic is made of long-lived FTP connections. The steady background traffic means that the throughput of TCP connections follows the "saw-tooth" curve [3]. This means that the connections see a steady loss rate [7]. The same is true for the CMTCP connections. When the loss rate is steady, the measurement interval does not significantly affect the accuracy of the measurement. We have also performed experiments with a bottleneck propagation delay set to 100ms. The results of simulations with 100ms delay (not shown here) are similar to that with 50ms delay, except that for low values of M, (e.g., $M \leq 2$ seconds) the friendliness ratio has large variance. This is due to the fact that as the round trip times increase, larger measurement intervals (i.e., larger values of M) are needed to obtain more reliable estimates of loss rates. These observations suggest that instead of using a fixed value for M, making M a fixed multiple of round trip time might result in better friendliness ratios. We discuss this issue further in Section 5.

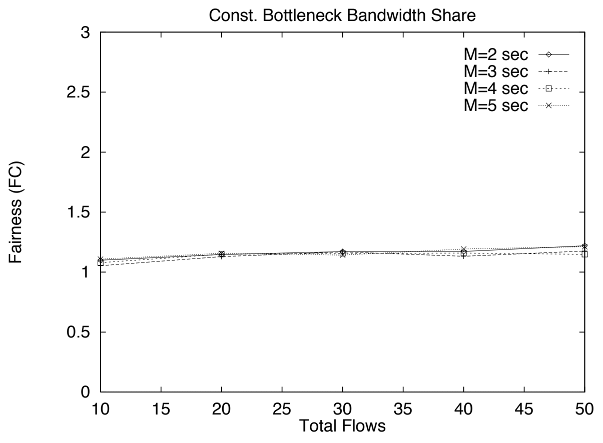
It is also important that the CMTCP connections achieve fairness amongst themselves. We define the ratio:

$$FC = \frac{\max_{1 \leq i \leq k_c} T_i^c}{\min_{1 \leq i \leq k_c} T_i^c}$$

to characterize the fairness achieved among the CMTCP connections. In Figures 3(a) and 3(b), we plot FC for various values of M and bottleneck link delay. Each data point is an average of three simulations. It can be seen that the CMTCP protocol achieves acceptable fairness among CMTCP connections in most cases.



(a) Bottleneck Delay 20ms



(b) Bottleneck Delay 50ms

Figure 3: Fairness (FC), Constant Bottleneck Bandwidth Share

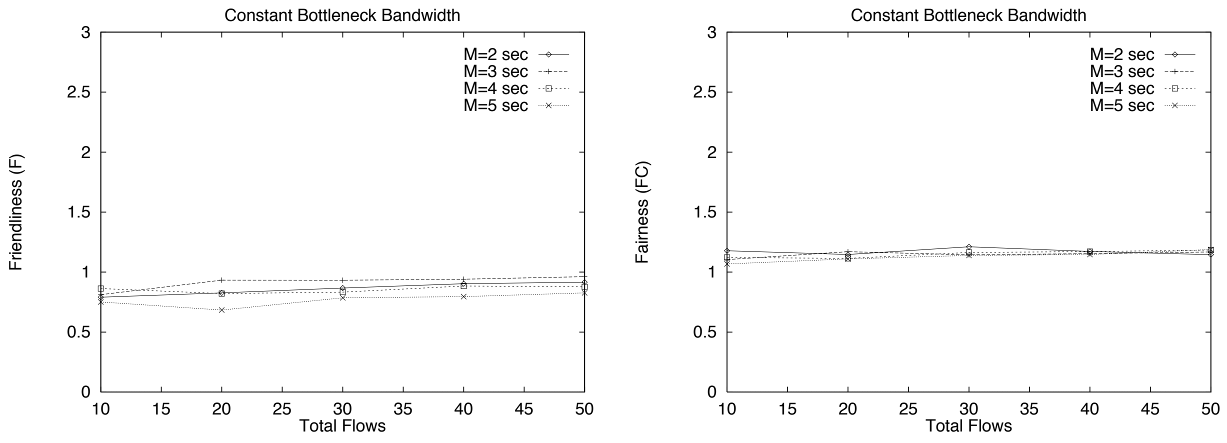
3.2 Constant Bottleneck Bandwidth

The setup for these simulations is similar to that in the previous subsection, except that the bottleneck bandwidth is held constant at 1.5Mbps. This roughly simulates a situation in which a number of connections share a T1 link. As the number of flows grows, the window sizes of individual TCP connections shrink, increasing the probability of timeouts. In such circumstances, the congestion control protocols proposed in [16, 19] are not be able to guarantee fairness. However, since CMTCP is based on a TCP characterization that takes timeouts into account, it *is* able to maintain acceptable levels of fairness, as can be seen from results reported in Figure 4(a). We also performed simulations with bottleneck delays set to 20 and 100 milliseconds. The results (not shown here) were similar to those results presented in the previous section for these bottleneck link propagation delays. In Figure 4(b) we that the CMTCP connections achieve acceptable fairness among themselves for the case of 50 ms propagation delays. The results for 20ms and 100ms bottleneck delays (not shown here) are similar.

3.3 Dynamically Arriving Medium-duration FTP Connections

In this simulation scenario, we study the effect of “slow” changes in the background traffic. Recall that in the simulations described so far, traffic consisted of infinite TCP and CMTCP connections. We now consider the case that there is one infinite-duration TCP connection, one infinite-duration CMTCP connection (i.e., $k = 2$), and additional traffic consisting of dynamically arriving TCP connections, each of which transfers a fixed amount of data. In computing F, we consider only the infinite-duration connections. The bottleneck link bandwidth is set to 1.5Mbps and the bottleneck delay is set to 50ms. The duration of simulation is 1000 seconds. The amount of data transferred by each background connection is chosen from a uniform distribution[4] with a mean of either 40KB or 80KB.

[4]$(0 \ldots 80KB)$ and $(0 \ldots 160KB)$, respectively.



(a) Friendliness (F) (b) Fairness (FC)

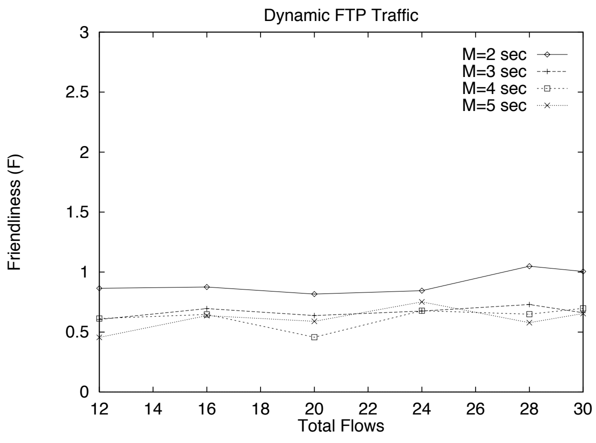
Figure 4: Constant Bottleneck Bandwidth Share, Bottleneck Delay 50ms

The interarrival times for the medium-duration FTP connections are chosen from a uniform distribution. We assume that all active flows get an equal share of the bottleneck bandwidth. Then, for a given average arrival rate, and a given average size of data transfer, we can calculate the average number of background connections that will be active at any given time using Little's law. A higher average number of background connections leads to more fluctuations in the background traffic, and in addition, the window size of each TCP connection tends to be smaller (due to a smaller bandwidth share), increasing the possibility of timeouts. We are interested in the performance of CMTCP protocol as the average number of background connections change. For plots in Figures 5(a) and 5(b) each connection transfers an average of 40KB and 80KB data, respectively.

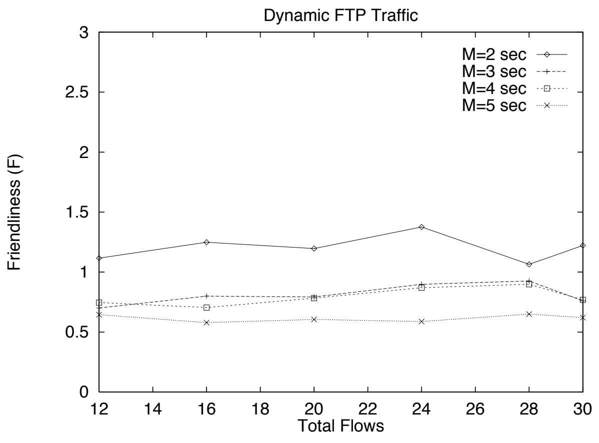
The results in Figure 5 show that CMTCP maintains a friendliness ratio of approximately one with a recomputation interval $M = 2$ seconds. The ratio decreases as the recomputation interval becomes larger. We conjecture that this behavior is due to the nature of the background traffic. As old connections terminate and new ones start, there are small periods of time during which the background traffic decreases slightly as the new connections go through their slow start phase. TCP is better able to take advantage of these small drops in the background traffic, due to its faster feedback mechanism. The CMTCP connection changes its sending rate only every M seconds, and hence is unable to take advantage of short-term drops in the background traffic.

3.4 ON/OFF UDP traffic

In this simulation scenario, we model the effects of competing web-like traffic (very small TCP connections, some UDP flows). It has been reported in [14] that WWW-related traffic tends to be self-similar in nature. In [25], it is shown that self-similar traffic may be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. Figure 6 presents results from simulations in which the "shape" parameters of the Pareto distribution



(a) Average Transfer Size 40KB



(b) Average Transfer Size 80KB

Figure 5: Friendliness (F), Dynamically arriving FTP connections

are set to 1.2. The mean ON time is 1 second and the mean OFF time is 2 seconds. During ON times the sources transmit with a rate of 12Kbps. The number of simultaneous connections is varied between 20 and 80. The simulation was run for 25000 seconds. As in the previous subsection, there are two monitored connections, an infinite TCP connection and an infinite CMTCP connection (i.e. $k = 2$). The bottleneck link bandwidth is set to 1.5Mbps and the bottleneck delay is set to 50ms. From the results in Figure 6, we can see that the protocol performs fairly well. The fairness index again decreases as the recomputation interval, M, increases. We conjecture that the reason for this is the inability of the CMTCP connection to take advantage of small periods of time in which the background traffic drops in intensity. Results for other values of shape parameter are similar.

4 Implementation and Experimental Results

As noted in [15], simulating an Internet-like environment is very difficult. It is thus essential to test protocols like CMTCP via implementation and experimentation in a real-world setting. Our goal here is thus to show that the approach is practical, and the performance of the protocol under real-world conditions is comparable to that observed in the simulations. We have implemented a prototype version of our protocol and tested it on several Unix systems. In this section, we first describe the implementation, and discuss some of the difficulties encountered. We then present the results from over 300 data transfers performed using this implementation.

4.1 Implementation

Our implementation of CMTCP is done in user space, on top of UDP. The sender side of CMTCP runs as two processes, one sending the data and the other receiving ACKs. The two processes communicate via shared memory. An earlier attempt to implement the protocol using multiple threads failed, as the

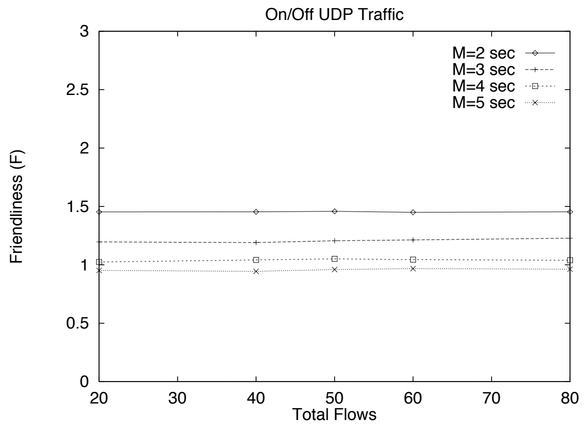


Figure 6: Friendliness(F), ON-OFF UDP Traffic, Shape = 1.2

p-threads package could not provide sufficiently accurate scheduling control to avoid starving either the sender or the receiver thread. We were able to reuse much of the `ns` simulation code for actual implementation. However, three important problems were encountered:

- A significant problem in any actual implementation is the the accuracy of the various timers involved. For simulation purposes, we could time out packets with arbitrary precision. This is not possible in actual implementation, as the timers are neither arbitrarily accurate nor are they free of overheads. In some of our early experiments we found that when using the `gettimeofday` and `select` system calls, we could not control inter-packet interval more accurately than within several milliseconds. While we could achieve better accuracy using busy waiting in the process that sent the packets out, this can possibly starve the process that receives ACKs. On a FreeBSD machine used in this study, busy waiting caused other problems that forced us to to use `gettimeofday` and `select` system call in our FreeBSD implementation. Due to these problems, it is not possible to clock packets out smoothly over the duration of each $round$, as mentioned in Section 2.2. Instead, we send packets out in small bursts. Consider round i. Let R_i be the round trip time estimate for this round, and r_i be the sending rate. The duration of the round is M time units. Then, the round is divided into bursts of duration R_i each. The number of bursts is thus, $b_i = M/R_i$. In each burst, n_i/b_i (rounded to nearest integer) packets are sent back-to-back, followed by a silence period of R_i time units. We conjecture that timer inaccuracies are responsible for at least 10% of the difference we see in the throughputs of TCP and CMTCP connections.
- Another important problem was the accuracy of the round trip times. The CMTCP protocol begins measuring the round trip time for each packet as soon as it is handed to the kernel socket using `sendto`. Thus, our round trip times include the time each packet spent waiting in the kernel buffers (similarly for ACKs). Thus, our estimate of round trip times is higher than that of the in-kernel TCP's.
- In our simulation studies, it was easy to ensure that the packet sizes for TCP and CMTCP con-

Hostname	Domain	Operating System
alps	cc.gatech.edu	SunOS 4.1.3
bmt	cs.columbia.edu	FreeBSD 2.2.7
edgar	cs.washinton.edu	OSF1 3.2
manic	cs.umass.edu	IRIX 6.2
maria	wustl.edu	SunOS 4.1.3
newton	nokia-boston.com	SunOS 5.5.1
sonic	cs.ucl.ac.uk	SunOS 5.5.1
void	cs.umass.edu	Linux 2.0.30

Table 1: Hosts used for empirical studies

nections were the same. It is more complicated to ensure this in practice. Our implementation currently does not employ any path MTU discovery algorithm [10], nor does it change the size of outgoing packets. For each experiment described in the next section, the packet size is held constant, determined by the MTU discovered by TCP in previous experiments between the same two hosts. While we have found that we seldom had problems with MTU value, it is hard to quantify the effects of constant packet size on throughput.

As a result of the implementation considerations noted above, we expect the results from implementation experiments to differ somewhat from the simulation experiments. However, we can still use the implementation to corroborate the intuition gained through our simulations, to examine CMTCP performance in real-world setting, and to provide a starting point for a more refined implementation.

4.2 Experimental Results

The hostnames, domains and operating systems of the machines used for the implementation study are listed in Table 1. To measure the fairness of CMTCP compared to TCP, we performed the following experiment. We established two connections between a pair of hosts. One of these connections was controlled by the CMTCP protocol, while the other was controlled by the TCP protocol. Both connections ran simultaneously, and transferred data for 1000 seconds, as fast as possible. The length of the recomputation interval, M, for CMTCP connection was set to 3 seconds; the receiver's declared window size, W_{max}, was set to 100 packets; and the initial sending rate, r_0, was set to approximately 40 packets/second. The throughput of the two connections were measured in terms of number of packets transferred in these 1000 seconds. Let us denote these throughputs T_C and T_T respectively. Figures 7, 8 and 9 show the results when the senders were $void$, $manic$ and bmt respectively. Since we are not interested in trends along the x-axis, we use E as our performance metric. For each of the first five bars, the x-axis shows the receiver. To plot this graph, at least 15 experiments were performed between the sender and the receiver at random times during the day and night[5], and for each experiment the value

[5]Experiments with bmt as a sender were performed only during the day.

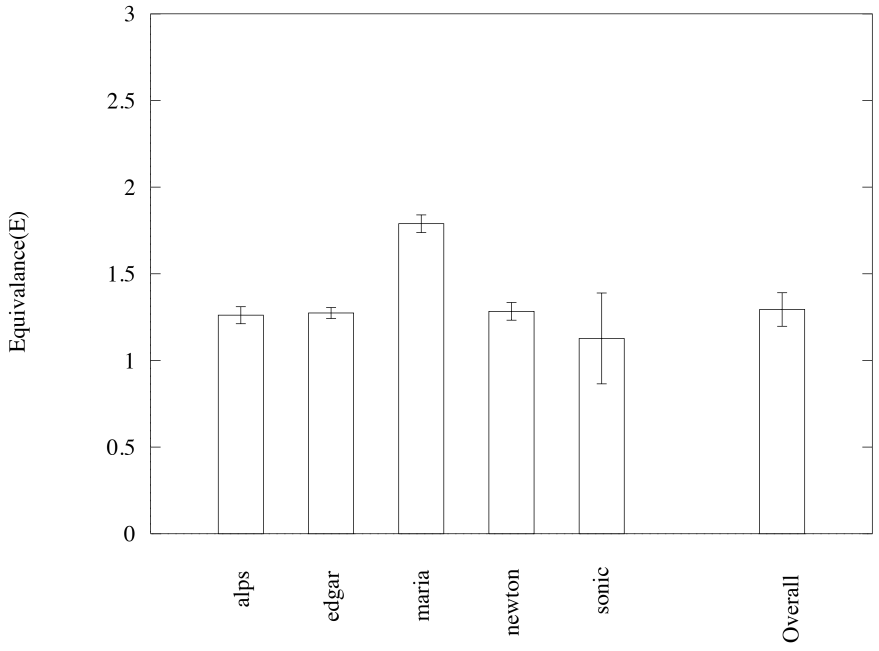


Figure 7: Experimental Results. Sender: void

of E was computed. It is suggested in [6] that data from such experiments should be represented by its median, and that the variation be represented by the semi-inter quartile range (SIQR), defined as half of the difference between the 25^{th} and the 75^{th} percentiles of the data set. Thus, the height of each bar is the median of that data set, while the the bar represents the SIQR, centered about the median. The last bar represents the median and the SIQR of all experiments.

It can be seen that in most cases, the CMTCP protocol achieves a throughput that is within 35-50$\%$ of the TCP throughput and that the difference seldom exceeds 75$\%$. The median of all three data sets taken together is 1.448 and the SIQR is 0.275. There are many possible reasons for the observed difference between the TCP and CMTCP throughputs. Some variation is unavoidable. We have found that the throughput of two simultaneous TCP connections between the same hosts can differ by as much as 10$\%$. Additional variation results from the various implementation difficulties described earlier. And finally, one must remember that the formula described in [12] is only an approximation.

Figures 7-9 are based on throughputs that have been computed over the entire duration of the experiment (i.e., 1000 seconds). It is also interesting to compare the difference in TCP and CMTCP as a function of time, and over shorter intervals of time. Such a comparison illustrates how well the CMTCP protocol performs at various time scales. In Figure 10, we plot the throughput of the CMTCP and the TCP connections between $manic$ and $edgar$, measured every 6, 12, 24 and 48 seconds respectively. It can be seen that CMTCP tracks variations in throughput of the TCP connection at quite well, at various time scales.

To measure the sensitivity of the protocol to the interval over which we measure the loss rate and update the sending rate (i.e. the value of M), we performed several data transfers between the same

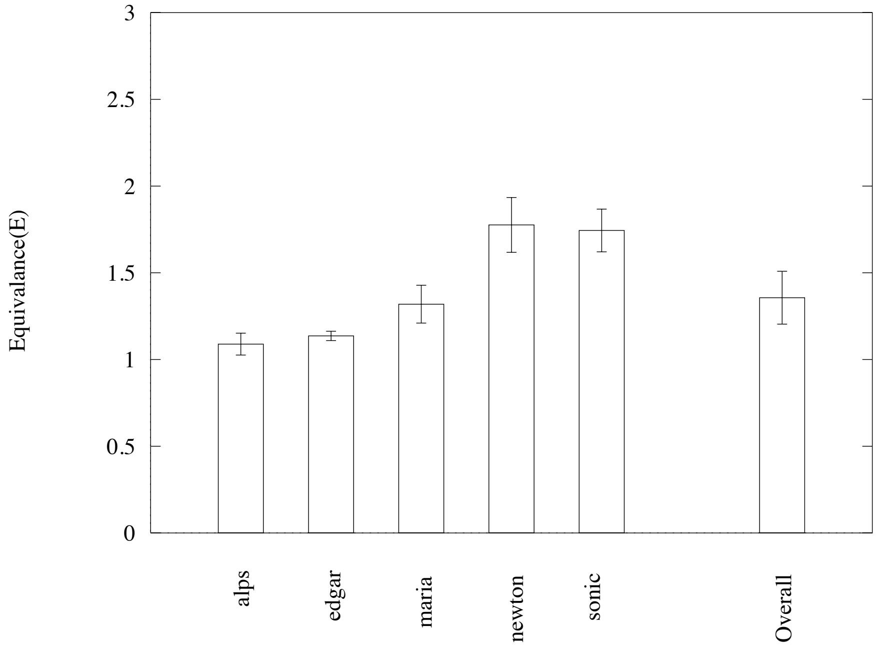


Figure 8: Experimental Results. Sender: manic

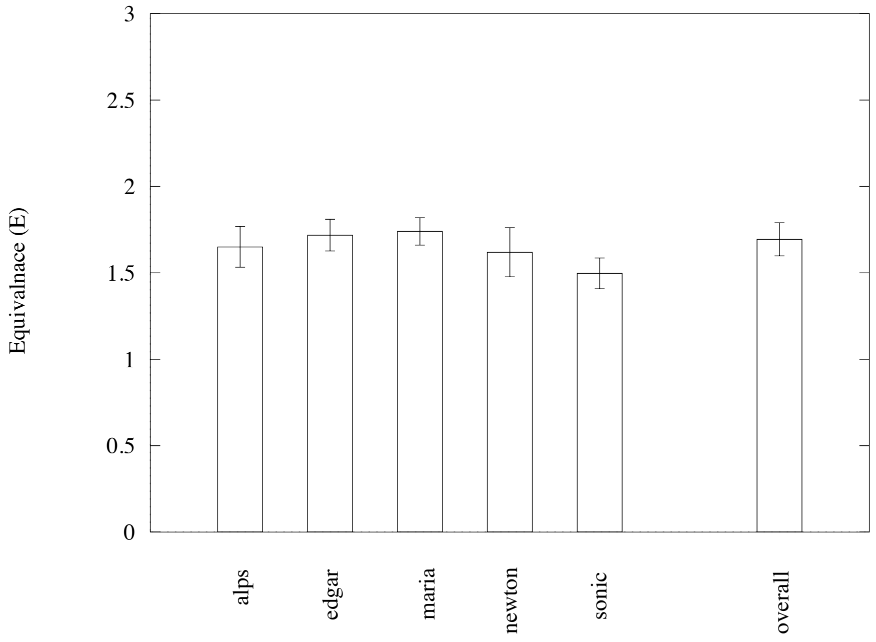
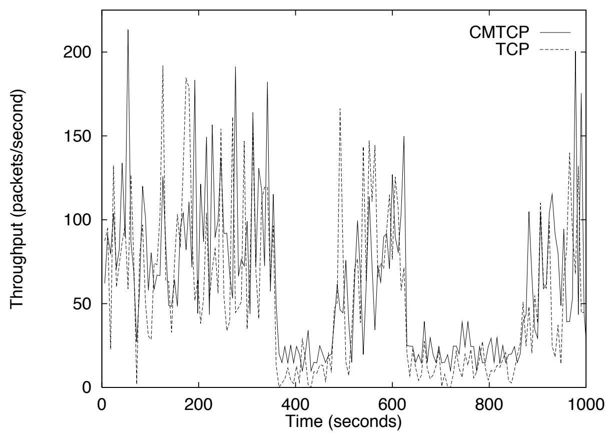
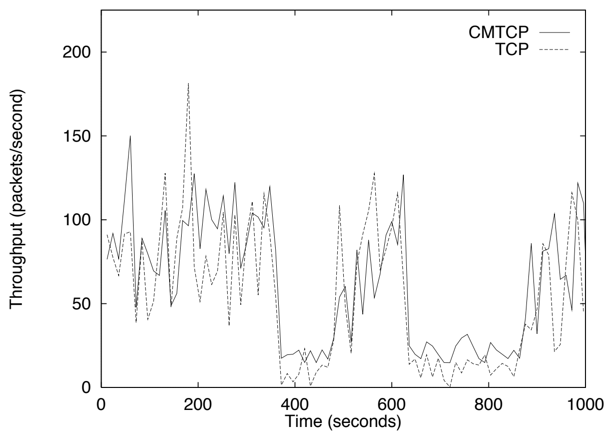


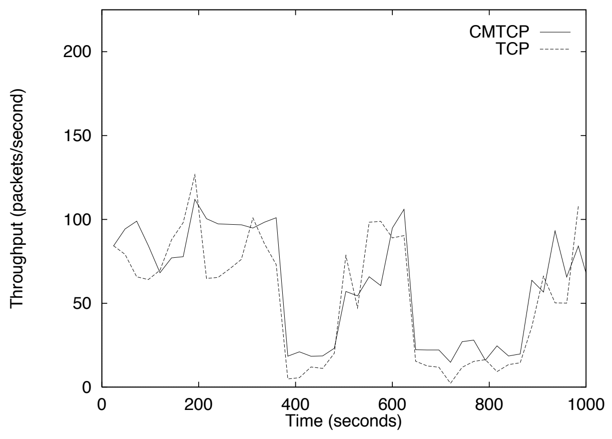
Figure 9: Experimental Results. Sender: bmt



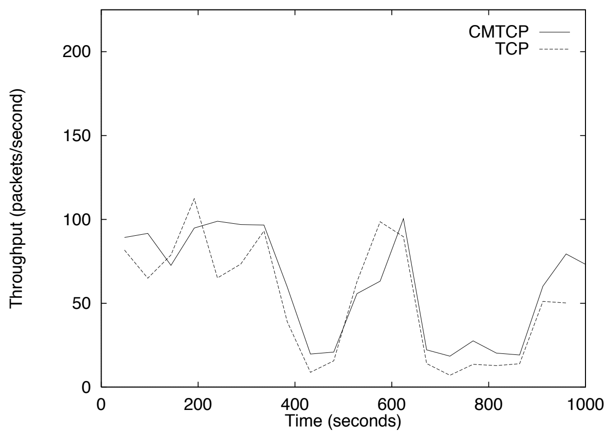
(a) Throughput every 6 seconds



(b) Throughput every 12 seconds



(c) Throughput every 24 seconds



(d) Throughput every 48 seconds

Figure 10: Throughput at various timescales

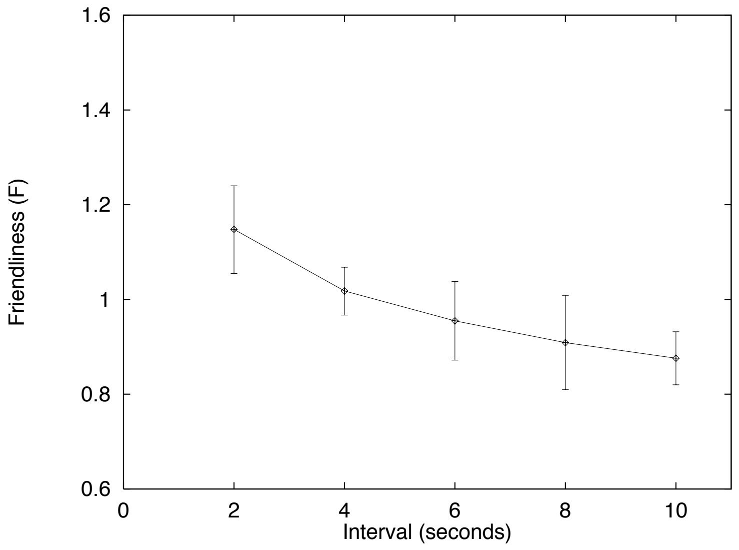


Figure 11: Sensitivity to measurement interval. void-alps

sender-receiver pair, using different measurement intervals. We now use F as our fairness metric, as we are interested in the trend in the performance metric as we vary M. In Figure 11 we show the results of one such study, performed between $void$ and $alps$. The measurement interval was varied between between 2 and 10 seconds. For each value of measurement interval, at least 20 experiment were conducted at random times. The data points are the medians of the throughput ratios, and the error bars represent the SIQR. One can see that as the measurement interval grows larger, the CMTCP protocol becomes less aggressive. This result is consistent with the simulation results presented in the previous section. From the results presented in this section, we can conclude that the protocol indeed performs well in a real world setting, despite the limitations and difficulties imposed by various implementation problems.

5 Discussion of Protocol Features

In this section we discuss the impact of some of the design choices made while simulating and implementing CMTCP.

As mentioned in Section 2, one of the most important challenges in using a formula-based approach in CMTCP is how to accurately estimate loss rates. In the protocol presented in this paper, we measure loss rates over fixed intervals of time (i.e., fixed M). There are several other possible choices. One could measure loss rates over a fixed number of samples; one could measure loss rates over a period of time that is some fixed multiple of round trip time estimate; one could choose a combination of these strategies: measure loss rate over a fixed number of samples or fixed time intervals, whichever occurs earlier. The main question is how to obtain loss rates estimates with tight confidence intervals, within a reasonable amount of time. This is a difficult problem and we are currently investigating this issue further. Measuring loss rates at fixed intervals of time serves as a baseline policy and it is of interest to see how well CMTCP performance with a simple loss estimation policy. CMTCP performance can now be further refined using better techniques for loss rate estimation.

Recall that CMTCP doubles its sending rate when no packets are lost in an entire recomputation period, since the formula in (2) is not valid for zero loss rate. During periods of no loss, the TCP window grows linearly (ignoring the initial slow start period). Since the sending rate is proportional to the window size, one can say that the sending rate of TCP grows linearly during periods of no loss. However, we found that if we try to mimic this behavior in CMTCP, the protocol performed poorly (i.e., the friendliness ratio was higher). This is due to the fact that in most of our simulations and Internet experiments, the fair share of the CMTCP connection tended to be relatively small. If, after a recomputation interval of no loss, we increased the rate in a linear fashion, the relative change in the rate was very high. This led to very high losses in the next round, which in turn dropped the sending rate to a very low value, leading again to a no loss, or low loss, period. This oscillatory behavior was detrimental to the performance of the protocol. Doubling the sending rate seems to offer a good compromise between responsiveness (ramping up the sending rate quickly) and avoiding oscillatory behavior.

The value of W_{max} can significantly affect the throughput computed using the formula in (2) at low loss rates. While in the simulations studies it is easy to ensure that competing TCP and CMTCP flows had the same value for W_{max}, this is hard to ensure in practice. We note that this problem is inevitable whenever flow control is employed: two TCP connections experiencing same network conditions, but having different values for W_{max}, will have different throughputs.

Another design issue is how to set the initial value for r_0. For the simulation and implementation results reported in this paper, we set this value to approximately. 40 packets/second. As long as the recomputation interval M is small compared to the time over which the friendliness or equivalence is being measured, the value for r_0 has little impact on the performance of CMTCP.

As mentioned in Section 4, timer inaccuracies and overheads force us to send packets out in small bursts, instead of clocking them out evenly over the duration of each $round$. The impact of this burstiness on performance of CMTCP protocol is hard to quantify. On one hand, one may imagine that burstiness would lead to slightly higher loss rates for CMTCP connection, forcing the throughput down. On the other hand, traffic from a TCP flow is somewhat bursty as well [3]. Thus the impact of bursty nature of CMTCP flow on friendliness ratio is hard to judge.

It should be noted that the formula in (2) is not valid for certain network scenarios, such as TCP connections running over modem lines with large dedicated buffers [12]. This implies that the CMTCP protocol would not work well in these situations either. We are currently working on solutions to this problem. We also note that CMTCP reacts to changes in network conditions only every M time units (i.e. duration of recomputation interval). If the network traffic conditions change faster than this, the difference between the throughput of a TCP connection and a CMTCP connection experiencing similar network conditions may be significant. Under such dynamic conditions, obtaining accurate loss estimates and round trip times can be problematic. We note that in real-world testing, Figures 7- 9, we have found that the protocol works well with a recomputation interval of three seconds.

6 Conclusions and Future Work

In this paper we have presented a TCP-friendly rate adjustment protocol for continuous media flows over best effort channel. The protocol achieves TCP-friendliness by changing its sending rate, corresponding to measured loss rate and round trip times, using TCP characterization developed in [12]. In addition to studying the protocol through simulations, we implemented a prototype version of the protocol and tested it with experiments over the Internet. The results of both simulation and implementation experiments show that the protocol is able to achieve throughputs that are close to the the throughput of a TCP connection traveling over the same network path.

Our results show that formula-based feedback-loop approach to congestion control and achieving TCP-friendliness is indeed practical. This work forms the basis for designing a comprehensive protocol for congestion control for CM flows that takes into account the timeliness requirements of CM data, as well as the need to be fair to background traffic.

We have identified several avenues for future work. Accurate estimation of packet loss rates is a difficult problem, and the accuracy significantly impacts the performance of the CMTCP protocol. We are currently working on developing better techniques for loss rate estimation. We plan to refine the implementation of the protocol, especially the implementation of various timers. We also plan to investigate if any other throughput formulas can be used in the feedback loop, and their impact on performance of the protocol. Above all, we are working towards developing a comprehensive protocol for congestion control of continuous media flows. The protocol will take into account the effects of limited buffer space available at the sender and the receiver, the timeliness requirements of the CM data, and loss tolerance of the specific media being sent.

References

[1] J. Bolot and A. Vega-Garcia. Control mechanisms for packet audio in the Internet. In *Proceedings IEEE Infocom96*, 1996.

[2] D. Clark and J. Wroclawski. An approach to sevice allocation in the internet. IETF draft-clark-diff-svc-alloc00.txt, July 1997.

[3] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3), July 1996.

[4] S. Floyd and V. Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1997.

[5] S. Jacobs and A. Eleftheriadis. Providing Video Services over Networks without Quality of Service Guarantees. In *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, 1996.

[6] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991.

[7] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. Note sent to end2end-interest mailing list, Jan 1997.

[8] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.

[9] S. McCanne and S. Floyd. ns-LBL Network Simulator, 1997. Obtain via http://www-nrg.ee.lbnl.gov/ns/.

[10] J. Mogul, C. Kent, C. Partridge, and K. McCloghrie. IP MTU Discovery Options. RFC1063, July 1988.

[11] T. Ott, J. Kemperman, and M. Mathis. The stationary behavior of ideal TCP congestion avoidance. in preprint.

[12] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of SIGCOMM'98*, 1998.

[13] C. Papadopoulos and G. Parulkar. Reatransmission-based error control for continuous media applications. In *Proceedings of NOSSDAV'96*, 1996.

[14] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols and self-similar network tarffic. In *Proceedings of ICNP'96*, 1996.

[15] V. Paxson and S. Floyd. Why we don't know how to simulate the Internet. In *Proccedings of the 1997 Winter Simulation Conference*, 1997.

[16] R. Rejaie, M. Handley, and D. Estrin. Rap: An end-toend rate-based congestion control mechanism for realtime streams in the internet. In preprint, 1998.

[17] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proceedings of SIGCOMM'98*, 1998.

[18] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. RTP: A Transport Protocl for Real-Time Applications. RFC 1889.

[19] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaption Scheme. In *Proceedings of NOSSDAV'98*, 1998.

[20] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC2001, Jan 1997.

[21] W. Stevens. *TCP/IP Illustrated, Vol.1 The Protocols*. Addison-Wesley, 1997. 10th printing.

[22] W. Stevens. *TCP/IP Illustrated, Vol.2 The Implementation*. Addison-Wesley, 1997. 4th printing.

[23] T. Turletti, S. Parisis, and J. Bolot. Experiments with a layered transmission scheme over the Internet. Technical report RR-3296, INRIA, France. Obtain via http://www.inria.fr/RRRT/RR-3296.html.

[24] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of INFOCOMM'98*, 1998.

[25] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-Similarity through high variability: Statistical Analysis of Ethernet LAN traffic at the source level. In *Proceedings of SIGCOMM'95*, 1995.

Appendix

$$f(W_{max}, p, R, B) = \begin{cases} \dfrac{\frac{1-p}{p} + W(p) + \frac{Q(p,W(p))}{1-p}}{R(W(p)+1) + \frac{Q(p,W(p))G(p)B}{1-p}} & \text{if } W(p) < W_{max} \\ \\ \dfrac{\frac{1-p}{p} + W_{max} + \frac{Q(p,W_{max})}{1-p}}{R(\frac{W_{max}}{4} + \frac{1-p}{pW_{max}} + 2) + \frac{Q(p,W_{max})G(p)B}{1-p}} & \text{otherwise} \end{cases}$$

where:

$$\begin{aligned} W(p) &= \frac{2}{3} + \sqrt{\frac{4(1-p)}{3p} + \frac{4}{9}} \\ Q(p,w) &= \min\left(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\right) \\ G(p) &= 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6 \end{aligned}$$