A Model Based TCP-Friendly Rate Control Protocol

Jitendra Padhye†, Jim Kurose†, Don Towsley†, Rajeev Koodli‡

†Dept. of Computer Science,
University of Massachusetts
Amherst, MA 01003
{jitu,kurose,towsley}@cs.umass.edu

‡Nokia Research Center,
3, Burlington Woods Drive,
Burlington, MA 01803
rajeev.koodli@research.nokia.com

Abstract— **As networked multimedia applications become widespread, it becomes increasingly important to ensure that these applications can co-exist 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 [9]. Using TCP for multimedia applications is not practical, since the protocol combines error control and congestion control, an appropriate combination for non-real time reliable data transfer, but inappropriate for loss-tolerant real time applications. In this paper we present a protocol that operates by measuring loss rates and round trip times and then uses them to set the transmission rate to that which TCP would achieve under similar conditions. The analysis in [13] is used to determine this "TCP-friendly" rate. This protocol represents a *first step* towards developing a comprehensive protocol for congestion control for time-sensitive multimedia data streams. We evaluate the protocol under various traffic conditions, using simulations and implementation. The simulations are used to study the behavior of the protocol under controlled conditions. The implementation and experimentation involve over 300 experiments over the Internet, using several machines in the US and UK. Our experimental and simulation results show that the protocol is fair to TCP and to other sessions running TFRCP, and that the formula-based approach to achieving TCP-friendliness is indeed practical.**

I. Introduction

Networked multimedia applications usually employ non-TCP protocols (usually UDP with some application level control) to transmit continuous media (CM) data such as audio and video. As these applications become widespread, it becomes increasingly important to ensure that they are able to co-exist with each other and with current TCP-based applications. A key requirement of such a co-existence is the implementation 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 data at the rate at which it was encoded, regardless of the congestion state of the network.

Two major considerations come into play when designing a congestion control protocol for CM applications. First, because these 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 [14, 18]. Second, since the applications must co-exist with TCP-based applications, the congestion control algorithms should adapt their rate in a way that "fairly" shares congested bandwidth with TCP applications. One definition of "fair" is that of TCP "friendliness" [9] – if a non-TCP connection shares a bottleneck link with TCP connections, traveling over the same network path, then the non-TCP connection should receive the same share of bandwidth (i.e., achieve the same throughput) as a TCP connection.

To develop a comprehensive CM congestion control protocol, one can begin by designing a congestion control protocol that sets the transmission rate in a TCP-friendly manner. Once such a "strawman" or baseline protocol is designed, it can then be modified to support the timeliness requirements of CM data, perhaps with some loss of "friendliness." The design of the "strawman" TCP-friendly protocol must be flexible enough to allow such modifications. This requirement for flexibility rules out the use of TCP itself as the baseline protocol. The congestion control mech-

This material is based upon work supported by the National Science Foundation under Grant Nos. CDA-9502639 and NCR-9508274. Part of the work was done when the first author worked for Nokia Research.

anisms of TCP are tightly coupled with the mechanisms that provide reliable delivery, an appropriate combination for non-real time reliable data transfer, but inappropriate for loss-tolerant time-sensitive CM applications. In this paper, we propose a simple baseline TCP-friendly rate control protocol (TFRCP) that does not couple error-recovery and congestion control, and retains sufficient flexibility for later modifications.

We present a congestion control algorithm that controls the sending rate in a manner that is roughly equivalent to that of TCP. Specifically, if a TCP connection achieves throughput X under given network conditions and measured over a given interval length, then the proposed protocol should also have a throughput of X over an interval of the same length and under the same network conditions. Note that the throughput X has to be measured over some time interval, and based on the definition of "TCP-Friendliness" proposed in [9], we assume that this interval is significantly larger than the round trip time. The actual transmission rate, X, is determined by using a model-based characterization of TCP throughput in terms of network conditions such as mean round trip time and loss rate. We base our protocol on the model proposed in [13]. In [23] the authors have proposed a similar approach for multicast congestion control, using the formula proposed in [9]. Our protocol differs form theirs in that we use a more accurate characterization of TCP and unlike [23] we do not require the use of data layering. Other TCP-friendly baseline protocols that try to mimic the major features of TCP congestion control algorithm without providing reliable delivery have been proposed [7, 17, 20, 24]. Some ongoing work, based partially on our ideas, with a focus on formula-based multicast congestion control, is also reported in [5, 6]. We discuss some of these protocols and their limitations in the next section.

We believe there are several advantages to taking a formula-based approach towards developing a TCP-friendly congestion control scheme. First, a formula based approach is *flexible*. By changing the formula, one can easily adjust the performance of the protocol. This feature can later be exploited for making the protocol sensitive to the timeliness requirement of the media being transported. In addition, if TCP and non-TCP flows are treated separately in the network (perhaps using a scheme such as [2]), then the formula-based approach can be modified to allow non-TCP flows to compete only against one another. Finally, 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.

We evaluate the protocol under various traffic conditions, using simulations and implementation. The simulations are used to study the behavior of the protocol under controlled conditions. The implementation and experimentation involve over 300 experiments over the Internet, using several machines in the US and UK. Our experimental and simulation results show that the protocol is fair to TCP and to other sessions running TFRCP, and that the formula-based approach to achieving TCP-friendliness is indeed practical.

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

II. Rate Adjustment Protocols

Several TCP-friendly rate adjustment protocols have recently been reported in the literature [7, 17, 20, 23, 24]. Of these, [23, 24] are specific to multicast applications, while [7, 17, 20] are unicast oriented. We now briefly review each of these five schemes, describe the new TFRCP protocol, and show how it overcomes some of the limitations of earlier work.

A. Previous Work

In [7], authors describe a protocol that may be classified as a "TCP-Exact" approach. They propose a protocol which manages its window size in exactly the same way as TCP does, but instead of retransmitting lost packets, it allows the user to send new data in each packet. The principle concern with this protocol is its inflexibility. Since the protocol strictly adheres to TCP window dynamics, it would be hard to modify it to take into account timeliness requirements of CM data delivery.

The TCP-friendly protocols reported in [17, 20, 23, 24] are based (either explicitly or implicitly) on the TCP characterization first reported in [9] and later formalized in [10, 12]. This characterization states that in absence of timeouts, the steady state throughput of a long-lived TCP connection is given by:

$$\text{Throughput} = \frac{C}{R * \sqrt{p}} \quad (1)$$

where C is a constant that is usually set to either 1.22 or 1.31, depending on whether or not receiver uses delayed acknowledgments, 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 [10]. The packet loss rate provides an *upper bound* on the value of p, and may be used as an approximation. The key assumption behind the characterization in (1) is that timeouts do not occur at all. Consequently, it is reported in [10] 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 [10, 13] 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. 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). Apart from not being TCP-friendly at loss rates above 5%, both schemes rely on data layering, which is not easy to achieve 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 [20] the authors propose a scheme that is suitable mainly for unicast applications, but may be modified for multicast applications. The scheme relies on regular RTP/RTCP reports [19] sent between the sender and the receiver to estimate the 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 [17] 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.

Although the protocols reported in [20] and [17] do not explicitly use (1) to control their rates, the work in [9, 10, 12] has shown that the relationship between loss rate and the throughput of these protocols will be similar to (1). As a result, these protocols will not be "TCP-friendly" at loss rates higher than 5%. While [20] ignores this problem, in [17] 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 [21] 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.

B. The TFRCP Protocol

The TFRCP protocol is a rate-adjustment congestion control protocol that is based on the TCP characterization proposed in [13]. Unlike [9, 10, 12], the characterization in [13] 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, which tend to produce correlated losses. If a TCP-Reno connection encounters correlated losses, it tends to experience a significant number of timeouts [3]. In [13] 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 [21]. A complete statement of the formula is presented in the Appendix.

There are two parts to the TFRCP protocol: 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. Note that each packet is ACKd eight times, providing some protection against ACK losses.

Let us now consider the sending rate computation in detail. Consider round i. Let r_i be the sending rate for this round, R be the the current round trip time estimate, and B be the estimate of the base timeout value. 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[1]. 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 ACKd 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.

[1]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 IV.

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

- Otherwise, $y_i \neq 0$. Let $p_i = \frac{y_i}{x_i + y_i}$. In this case, the rate for round $i+1$ is

$$r_{i+1} = f(W_{max}, R, p_i, B)$$

where f is defined in (2). It is here that the analytic characterization in [13] comes into play.

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 initial values of R and B are set in a manner similar to TCP [22].

TFRCP has no built-in error recovery mechanisms. When a comprehensive congestion control protocol, based on TFRCP is developed, the applications will be able 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" [17]. 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 TFRCP protocol uses the model proposed in [13], which takes into account the effect of timeouts and automatically reduces the sending rate to very small values at high loss rates.

The key question is how frequently the sender should re-compute the rate, i.e., how to determine the value of M. In the following section we use simulations to explore various strategies for choosing M, and their impact on the performance of the protocol.

III. Simulation Results

In this section we present simulation studies of the TFRCP protocol. The simulations are used to study the behavior of the protocol under controlled conditions. In the following section we present additional studies carried out over the Internet. We have used the `ns` simulator [11] for our simulations. There are two main challenges for any simulation study of this nature: first, how to select appropriate network topologies and how to effectively model the background traffic and second, how to define and measure appropriate performance metrics. Several difficulties in this regard are pointed out in [16]. Thus, before we present any simulation results, we discuss our simulation topology and our performance metrics.

A. Simulation Topology

In our simulations, we use a simple topology to uncover and illuminate the important issues; our experiments with TFRCP 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, 17, 20] have used similar topologies.

The problem of accurately modeling background traffic is more difficult. We consider three types of background traffic: infinite-duration FTP-like connections, medium-duration FTP -like connections and self-similar UDP traffic. The infinite-duration FTP

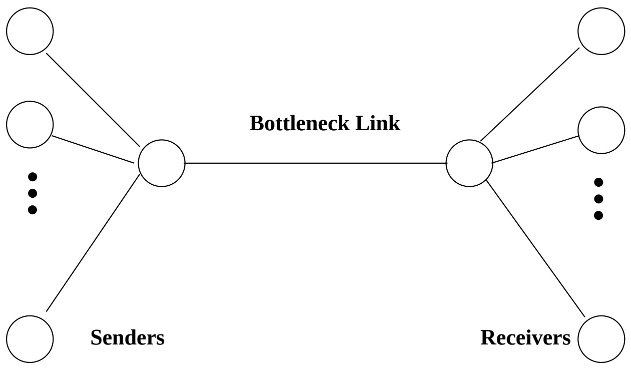


Fig. 1. Simulation Topology

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 [15, 25].

When multiple TCP connections are simulated over a single bottleneck link, the connections can become synchronized. 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 TFRCP connections.

B. Performance Metrics

Recall that we view TFRCP protocol as only a *first step* towards developing a comprehensive congestion control protocol for CM data flows. Thus, we are only interested in measuring the "TCP-friendliness" of the TFRCP protocol. We define the "friendliness" metric as follows. Let k_c denote the total number of monitored TFRCP connections and k_t denote the total number of monitored TCP connections. We denote the throughput of the k_c TFRCP 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:

$$T_C = \frac{\sum_{i=1}^{k_c} T_i^c}{k_c} \qquad \text{and} \qquad T_T = \frac{\sum_{i=1}^{k_t} T_i^t}{k_t}$$

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

$$F = T_C/T_T$$

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

$$E = \max(T_T/T_C, T_C/T_T)$$

Note that the value of E is always ≥ 1. E gives a better visual representation of the closeness of the throughputs achieved by 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 are interested in *trends*, and use E otherwise. It is also important that the TFRCP 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 TFRCP connections.

C. Simulation Scenarios

We now present results of performance evaluation of TFRCP protocol in various simulation scenarios.

C.1 Long duration flows with constant bottleneck bandwidth

In this scenario we consider traffic made up entirely of equal numbers of infinite-duration TCP connections and infinite-duration TFRCP connections. All connections 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 steady state behavior of TFRCP protocol. If TFRCP performs well (i.e., in a TCP-friendly manner), the TCP and TFRCP connections should see approximately the same throughput.

We vary the total number of flows in the network between 10 and 50. Half of these connections are TCP connections and the rest are TFRCP connections. The initial sending rate, r_0, for all TFRCP connections was set to approximately 40 packets/second. The bottleneck bandwidth is held constant at 1.5Mbps, and the bottleneck delay is set to 50ms. 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 [17, 20] are not be able to guarantee fairness.

We consider three different ways to determine how frequently TFRCP should recompute its rate:

• Fixed recomputation interval, i.e. we use a fixed value for M. We call this strategy S1.

• The recomputation interval is a fixed multiple of round trip time. If at the beginning of round i the round trip time is rtt_i, then the next recomputation is performed after $K * rtt_i$ time units, where K is constant. We call this strategy S2.

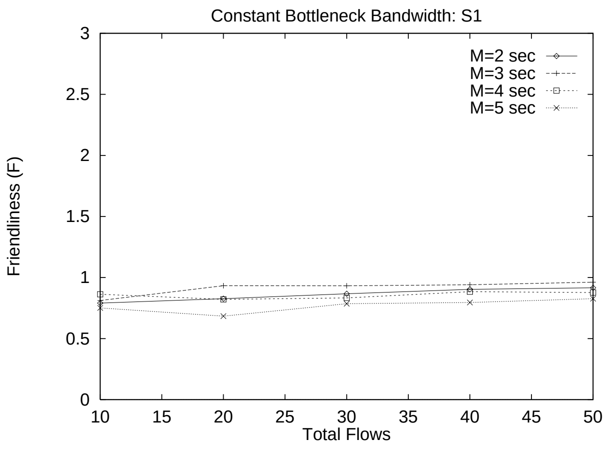
• The recomputation interval is calculated at the beginning of each round, and is set to sum of two numbers, one of which is a constant while the other is chosen from a uniform random distribution. This strategy will further prevent TFRCP connections from synchronizing with each other. We call this strategy S3.

In Figure 2(a) we present simulation results for the case in which the TFRCP protocol uses strategy S1, with five values of M 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.

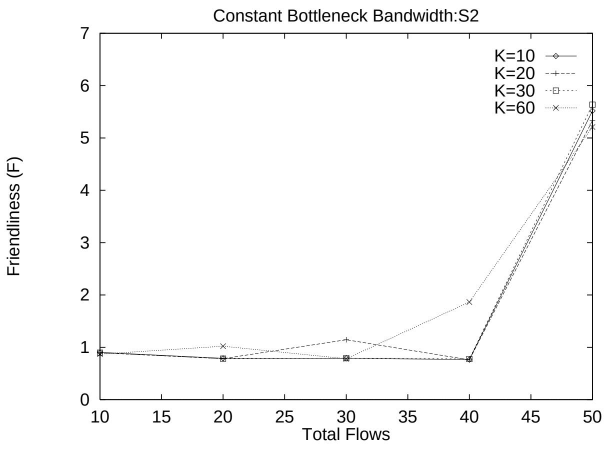
In Figure 2(b) we present simulation results when TFRCP protocol uses strategy S2, with four values of K between 10 and 60. We notice that as the load on the network increases, the resulting TFRCP behavior is more aggressive than TCP. As the load on the network increases, the round trip time experienced by each flow also increases. As a result, each TFRCP flow re-computes its rate less frequently. TCP reduces its transmission rate multiplicatively every time it encounters a loss, and increases it only additively in case of no loss, thus the slowness of response of TFRCP flows to react to losses hurts the throughput of TCP connections. Thus TFRCP is more aggressive, and clearly S2 is not an appropriate strategy for deciding recomputation intervals.

In Figure 2(c) we present simulation results where TFRCP protocol uses strategy S3. For each line we use a different constant and a different uniform random distribution: $0.3 + [0, 5.4], 1.5 + [0, 3]$ and $2.7 + [0, 0.6]$. For this simulation study, all TFRCP connections were started simultaneously. It can be seen that in this third case the protocol is able to maintain a friendliness ratio close to 1.

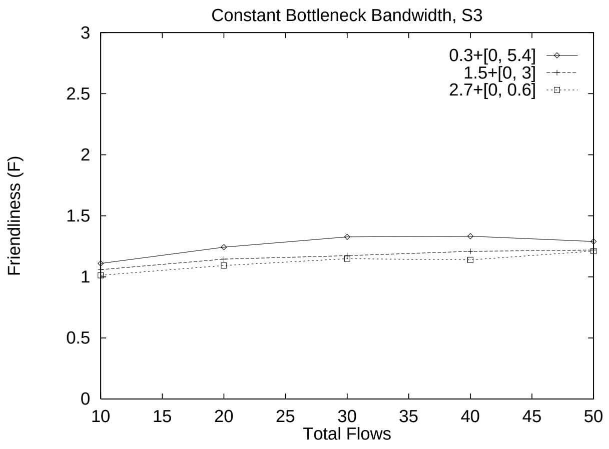
We have performed simulations with other bottleneck delays and observed similar results. In the rest of this section we only present results using strategy S1. We do this for two reasons. First, strategy S1 is the simplest strategy. The goal of this paper is to



(a) Strategy S1



(b) Strategy S2



(c) Strategy S3

Fig. 2. Constant Bottleneck bandwidth, Bottleneck Delay 50ms

present TFRCP protocol as a baseline policy; use of a simple policy to decide the recomputation interval is consistent with that goal. Second, the question of selecting the appropriate recomputation interval requires more complex answers than the three simple strategies described here. The recomputation interval must be short enough to allow TFRCP to be responsive, while at the same time it must be large enough to allow the loss rate measurements to be meaningful. This question is currently under research [5, 6]. Thus, it is appropriate to restrict the baseline protocol described here to the simplest strategy.

Recall that the TFRCP connections should be fair to each other as well. In Figure 3 we plot the value of FC when the TFRCP protocol uses strategy S1. It can be seen that the TFRCP protocol achieves acceptable fairness among TFRCP connections in most cases.

C.2 Long duration flows with constant bottleneck bandwidth share

In this scenario, the traffic is made up of infinite-duration TCP connections and infinite-duration TFRCP connections. All connections start at the beginning of the simulation and last until the end. 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 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 [17] and exhibit the full range of behavior such as slow start and congestion avoidance. Each experiment is repeated

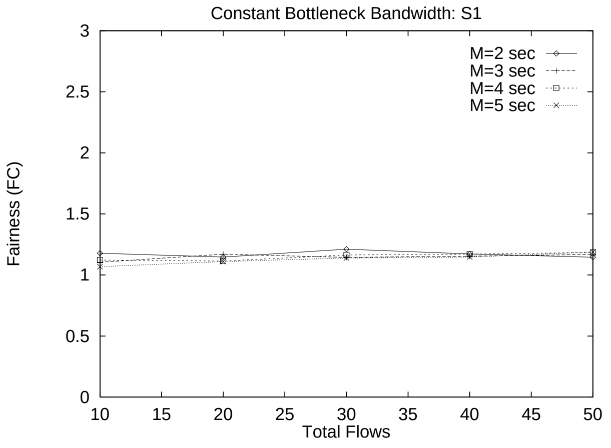


Fig. 3. Friendliness among TFRCP connections

for various values of the recomputation interval, M. The initial sending rate, r_0, for all TFRCP connections was set to approximately 40 packets/second. In Figures 4(a) and 4(b) we plot F and FC (TCP-friendliness and Fairness among TFRCP connections) for this scenario when the bottleneck delay was 50ms and the TFRCP connections used strategy S1. It can be seen that with steady state background traffic, the protocol is able to maintain a friendliness ratio close to 1, and the TFRCP connections are fair among themselves as well. We performed simulations with bottleneck delay of 20ms and 100ms as well (not shown here), and found that for small bottleneck delays, TFRCP 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. At higher round trip delays (100ms) the performance of the TFRCP protocol for small values of M (< 3 seconds) shows high variance, as the protocol is unable to gather sufficient samples to estimate loss rates accurately.

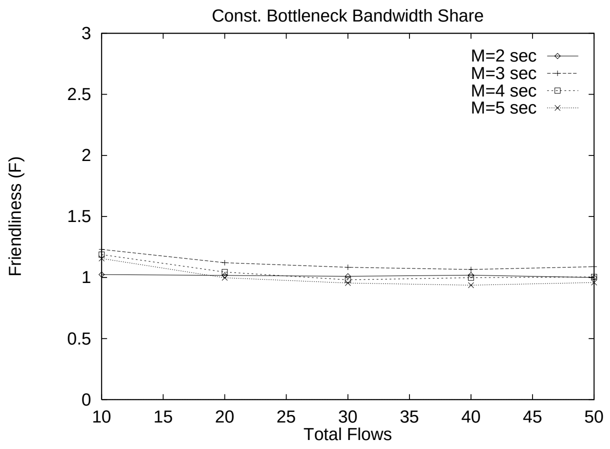
C.3 Dynamically Arriving Medium-duration FTP Connections

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 TFRCP connections. We now consider the case that there is one infinite-duration TCP connection, one infinite-duration TFRCP connection, 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 two 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. The interarrival times for the medium-duration FTP connections are chosen such that on average a constant number of background connections will be active. 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 TFRCP protocol as the average number of background connections change. For graphs in Figures 5(a) and 5(b) the data transferred by each connection is chosen from $[0, 80KB]$ (average 40KB) and $[0, 160KB]$ (average 80KB), respectively.

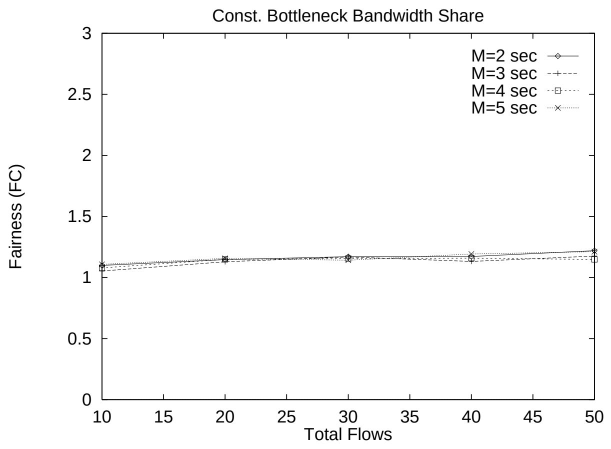
The results in Figure 5 show that TFRCP maintains a friendliness ratio of approximately one with a recomputation interval $M = 2$ seconds. The ratio decreases as the recomputation interval becomes larger. We conjecture that this behavior is due to the nature of the background traffic. As old connections 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 TFRCP connection changes its sending rate only every M seconds, and hence is unable to take advantage of short-term drops in the background traffic.

C.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 [15] 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" parameter of the Pareto distribution is 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 con-



(a) Friendliness (F)



(b) Fairness (FC)

Fig. 4. Constant Bottleneck bandwidth share, Bottleneck delay 50ms

nections, an infinite TCP connection and an infinite TFRCP 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 TFRCP protocol is still relatively fair. The fairness index again decreases as the recomputation interval, M, increases. We believe that this is due to the fact that the TFRCP connection recomputes its rate only after every M time units. Hence, it can not increase its sending rate during the small periods of time in which the background traffic drops in intensity. Results for other values of the shape parameter are similar.

D. Summary of simulation results

The simulations results presented in this section show that the TFRCP protocol is ‘TCP-friendly” under a wide variety of traffic conditions. We found that the strategy to use a fixed value for recomputation interval (M), works well for a wide variety of traffic conditions. While the simulation study is based on several different traffic scenarios, it is important to observe the performance of the protocol in real world. In the next section we discuss the implementation and present results based on experiments carried out over the Internet.

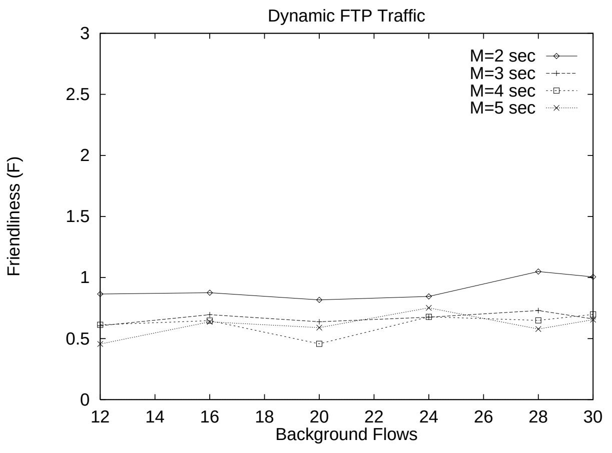
IV. Implementation and Experimental Results

As noted in [16], simulating an Internet-like environment is very difficult. It is thus essential to test protocols like TFRCP via implementation and experimentation in a real-world setting. Our goal here is thus to show that the approach is practical, and that 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 experiments performed using this implementation.

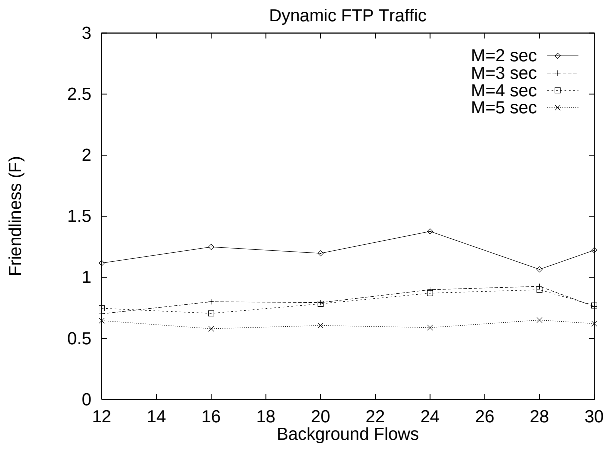
A. Implementation

Our implementation of TFRCP is done in user space, on top of UDP. The sender side of TFRCP 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 *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 the actual implementation. However, we encountered three important problems during the implementation:

• 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 an 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



(a) Average transfer 40KB



(b) Average transfer 80KB

Fig. 5. Friendliness (F), Dynamically arriving FTP connections

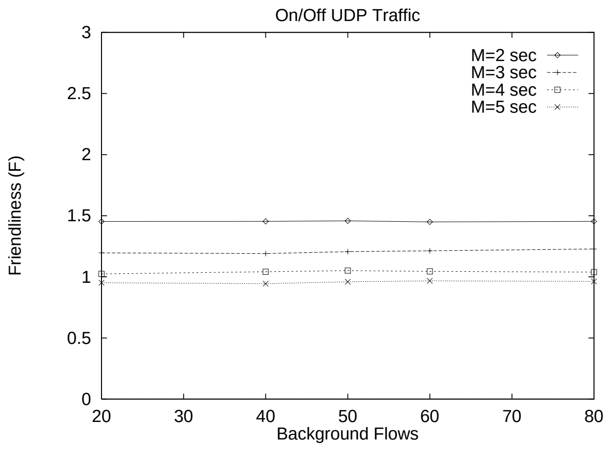


Fig. 6. Friendliness(F), ON-OFF UDP Traffic, Shape = 1.2

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 use `gettimeofday` and `select` system call in our FreeBSD implementation. Due to these difficulties, it is not possible to clock packets out smoothly over the duration of each *round*, as mentioned in Section II-B. Instead, we send packets out in small bursts. Consider round i. Let R be the round trip time estimate at the beginning of 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 each. The number of bursts is thus, $b_i = M/R$. In each burst, n_i/b_i (rounded to nearest integer) packets are sent back-to-back, followed by a silence period of R time units.

• Another important problem was the accuracy of the round trip times. The TFRCP 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 addition, due to additional difficulties with timers, we had to restrict the protocol to transmit *at least* one packet per round trip time.

• In our simulation studies, it was easy to ensure that the packet sizes for TCP and TFRCP connections were the same. It is more complicated to ensure this in practice. Our implementation currently does not

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 I
Hosts used for empirical studies

employ any path MTU discovery algorithm, 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 the 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 TFRCP performance in real-world setting, and to provide a starting point for a more refined implementation.

B. Experimental Results

The hostnames, domains and operating systems of the machines used for the implementation study are listed in Table I. To measure the fairness of TFRCP 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 TFRCP 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 the TFRCP 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 was measured in terms of number of packets transferred in these 1000 seconds. Let us denote these throughputs T_C and T_T respectively.

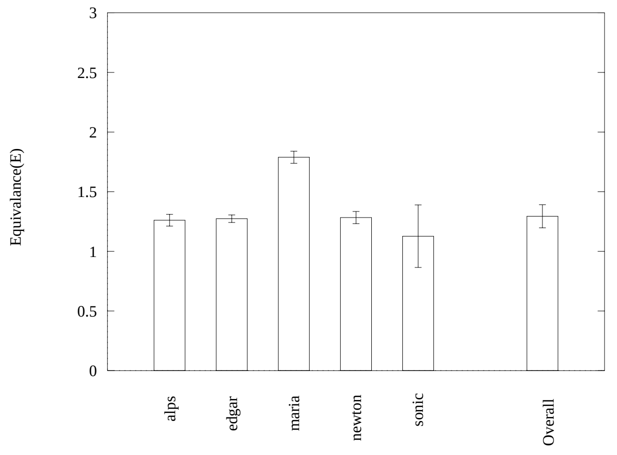
Figures 7(a)-7(c) show the results when the senders were *void*, *manic* and *bmt* respectively. Since we are not interested in trends along the x-axis, we use E as our performance metric. For each of the first five bars, the x-axis shows the receiver. To plot this graph, at least 15 experiments were performed between the sender and the receiver at random times during the day and night[2], and for each experiment the value of E was computed. It is suggested in [8] 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 25th and the 75th 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 TFRCP 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 TFRCP 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 [13] is only an approximation.

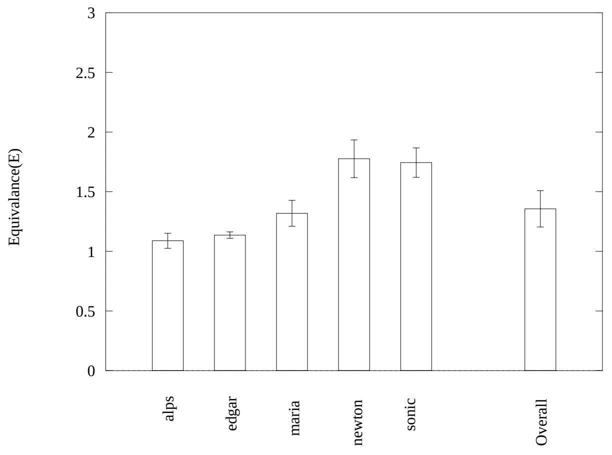
Figures 7(a)-7(c) are based on throughputs that have been computed over the entire duration of the experiment (i.e., 1000 seconds). It is also interesting to compare the difference in TCP and TFRCP as a function of time, and over shorter intervals of time. Such a comparison illustrates how well the TFRCP protocol performs at various time scales. In Figure 8, we plot the throughput of the TFRCP and the TCP connections between *manic* and *edgar*, measured every 6, 12, 24 and 48 seconds respectively. It can be seen that TFRCP tracks variations in throughput of the TCP connection 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 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

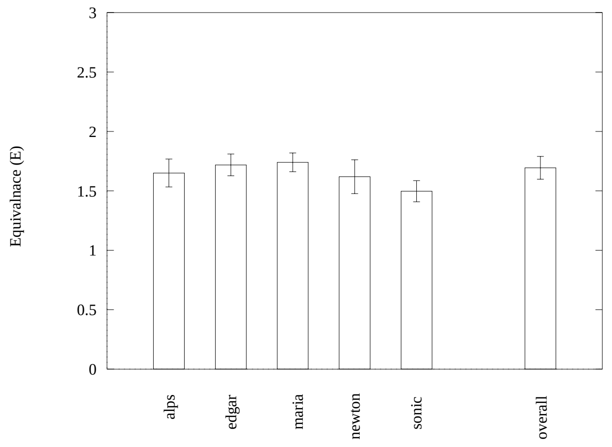
[2]Experiments with *bmt* as a sender were performed only during the day.



(a) Sender: void



(b) Sender: manic



(c) Sender: bmt

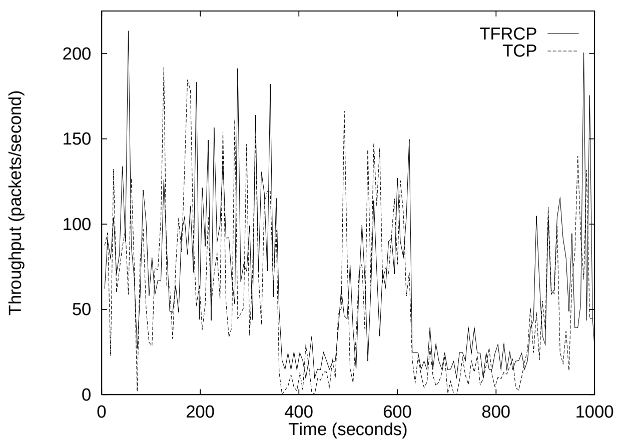
Fig. 7. Experimental Results

as we vary M. In Figure 9 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 TFRCP 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 issues.

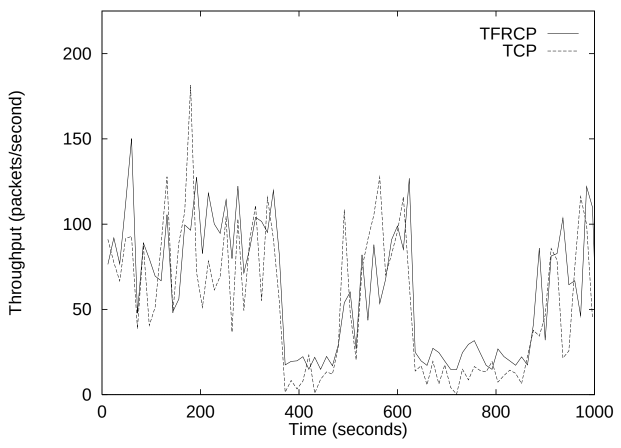
V. Discussion of Protocol Features

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

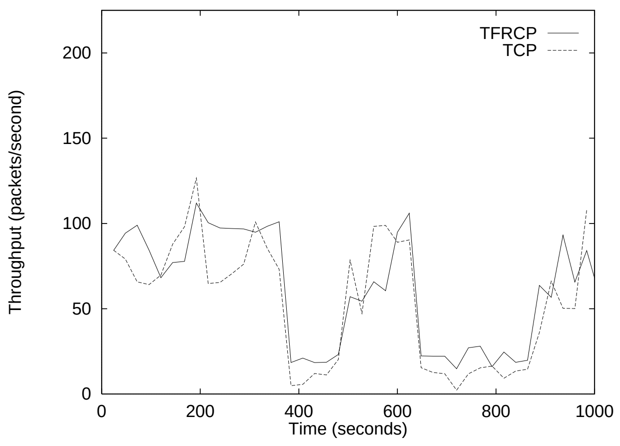
Recall that TFRCP doubles its sending rate when no packets are lost in an entire recomputation period, since the formula in (2) is not valid for zero loss rate. During periods of no loss, the TCP window grows linearly (ignoring the initial slow start period), by one every RTT. Since the sending rate is proportional to the window size, one can say that the sending rate of TCP grows linearly during periods of no loss. We found that when we try to mimic this linear increase behavior in TFRCP, 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



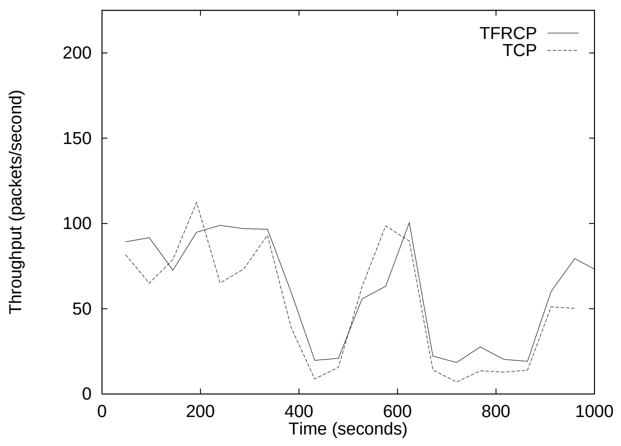
(a) Throughput every 6 seconds



(b) Throughput every 12 seconds



(c) Throughput every 24 seconds



(d) Throughput every 48 seconds

Fig. 8. Throughput at various timescales

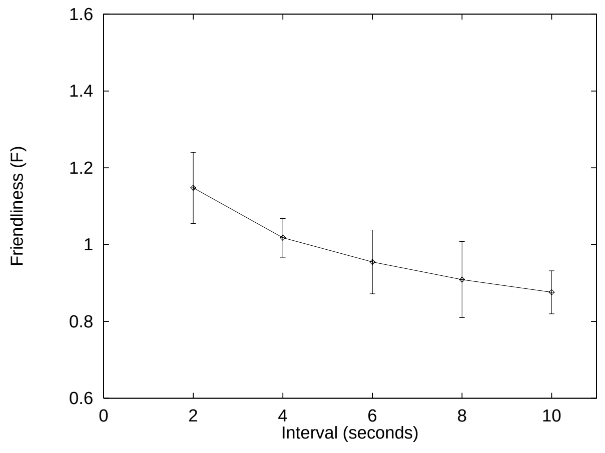


Fig. 9. Sensitivity to measurement interval. void-alps

experiments, the fair share of the TFRCP 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 (e.g., if no loss occurred during a three-second period, and if RTT was 100ms, the rate would increase by $3/0.1 = 30$). 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 TFRCP 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 was small compared to the time over which the friendliness or equivalence was being measured, the value for r_0 had little impact on the performance of TFRCP.

As mentioned in Section IV, 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 TFRCP protocol is hard to quantify. On one hand, one may imagine that burstiness would lead to slightly higher loss rates for TFRCP 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 TFRCP 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 [13]. This implies that the TFRCP protocol would not work well in these situations either. We are currently working on solutions to this problem. We also note that TFRCP reacts to changes in network conditions only every M time units (i.e. the duration of recomputation interval). If the network traffic conditions change on a faster time scale, the difference between the throughput of a TCP connection and a TFRCP 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(a)- 7(c), we have found that the protocol works well with a recomputation interval of three seconds. One may question if achieving TCP-Friendliness at large time granularities is useful at all. We would like to point out that multiple TCP connections going over the same network path need not achieve same throughput on a time scale comparable to the round trip time. Thus, fairness needs to be measured over time intervals longer than a few round trip times. One must also note that very short TCP connections such as HTTP transfers, do not achieve friendliness even among themselves. Hence, we have restricted ourselves to achieving fairness between long term TCP and TFRCP connections. We believe that as long as the duration of a flow is significantly larger than M, the TFRCP protocol achieves this goal.

VI. Conclusions and Future Work

In this paper we have presented a TCP-friendly rate adjustment protocol. The protocol achieves TCP-friendliness by changing its sending rate, based on TCP characterization developed in [13]. using the measured loss rate and round trip times 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. Thus, we conclude that formula-based feedback-loop approach to congestion control and achieving TCP-friendliness is indeed practical.

We have identified several avenues for future work. 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, along with the timeliness requirements 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] M. Handley and S. Flyod. Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC). Unpublished Manuscript. http://www.east.isi.edu/RMRG/newindex.html.

[6] M. Handley and S. Flyod. TCP-Friendly Simulations. Unpublished Manuscript.

[7] 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.

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

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

[10] 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.

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

[12] T. Ott, J. Kemperman, and M. Mathis. The stationary behavior of ideal TCP congestion avoidance. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps.

[13] 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.

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

[15] 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.

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

[17] R. Rejaie, M. Handley, and D. Estrin. An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proceedings of INFOCOMM 99*, 1999.

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

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

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

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

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

[23] T. Turletti, S. Parisis, and J. Bolot. Experiments with a layered transmission scheme over the Internet. Technical report RR-3296, INRIA, France.

[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

$$f(W_m, p, R, B) = \begin{cases} \dfrac{\frac{1-p}{p} + W(p) + \frac{Q(p,W(p))}{1-p}}{R(W(p)+1) + \frac{Q(p,W(p))G(p)B}{1-p}} & W(p) < W_m \\[2ex] \dfrac{\frac{1-p}{p} + W_m + \frac{Q(p,W_m)}{1-p}}{R(\frac{W_m}{4} + \frac{1-p}{pW_m} + 2) + \frac{Q(p,W_m)G(p)B}{1-p}} & W(p) \geq W_m \end{cases}$$

where:

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