

A TCP-friendly Protocol for Media Streams over Best Effort Networks

Hala ElAarag

Dept. of Math. & Computer Science
Stetson University
DeLand, FL 32720 USA
helaarag@stetson.edu

Mostafa Bassiouni

School of Electrical Eng. & Computer Science
University of Central Florida
Orlando, FL 32816 USA
bassi@cs.ucf.edu

Abstract

In this paper, we develop an analytical model for the congestion control mechanism of an Internet friendly transport-level protocol (IFTP). IFTP does not require routers to provide any additional mechanisms to support it. We evaluate IFTP through simulation and show that it solves the TCP-friendly problem. We also test IFTP against some important performance metrics such as packet delay and delay jitter.

1 Introduction

TCP's congestion control mechanisms [1] adapt TCP's throughput as a function of packet loss [2]. Some popular transport protocols, e.g. UDP, used for the transfer of multimedia applications do not incorporate any end-to-end congestion control. When TCP and UDP flows share a congested link, the TCP flows reduce their sending rate in response to congestion, while UDP flows uncooperatively use the available bandwidth. UDP is considered "non TCP-friendly".

In this paper, we develop a protocol that controls the sending rate of best-effort flows in a manner that is roughly equivalent to TCP. The transmission rate is determined based on an analytical model that emulates TCP throughout the lifetime of a connection. It takes into consideration network conditions such as round trip times and packet losses. Our protocol then computes a TCP-friendly sending rate by adjusting the inter-packet gap. The protocol is easy to implement, has little overhead, and guarantees fairness by preventing non-TCP flows from usurping the Internet.

The rest of the paper is organized as follows. Section 2 presents some of the previous work that addressed the TCP-friendliness problem and proposed solutions. In section 3, we present the analytical model which is the basis of the TCP-friendly protocol proposed in this paper. We discuss the implementation of IFTP in section 4. Performance evaluation of the IFTP protocol using simulation tests is presented in section 5. Finally, section 6 gives conclusions of the paper.

2 Related Work

The earliest discussion of the TCP-friendly problem was done by Floyd in her work with Mahdavi [3] and Fall [4]. The closest to our work is the Rate Adaptation Protocol (RAP) [5]. However, RAP uses AIMD mechanism to emulate TCP. AIMD is a steady state model that can be reached in networks with low loss rates [3]. This means that RAP emulates TCP only in the steady state. Thus only long file transfers suffering low loss rates can benefit from RAP. IFTP on the other hand mimics TCP in all its stages. This makes IFTP suitable for both short and long file transfers. Also IFTP would perform better than RAP in networks with high loss rates where TCP behavior is far from AIMD behavior [6]. Rejaie et al. in [5] provide simulation results that tests RAP's TCP-friendliness only. In this paper, we provide results that test IFTP's TCP-friendliness as well as some of the important metrics such as packet delay, delay jitter, packet loss and link utilization.

3 Analytical Model

In this section, we derive the conditions for constructing a rate-based control mechanism for IFTP that gives a faithful emulation of TCP behavior in all stages of the lifetime of the transport-level connection.

Consider two concurrent connections in a WAN environment: an unreliable connection running the IFTP transport protocol for real-time media streams and a reliable connection running TCP. The general TCP behavior used in the design of IFTP can be described as follows. TCP starts its connection in the slow start phase. It doubles the number of packets in its congestion window (*cwnd*) every round trip until it reaches a threshold called *ssthresh*. It then enters the congestion avoidance phase where it increases its window linearly until it reaches the maximum receiver window. If a number of duplicate acknowledgments (ACKs), usually three, are received, TCP assumes that a packet is lost. TCP then reduces the size of the sender's window by half and enters the congestion avoidance phase.

3.1 Inter-packet gap:

First we calculate the inter-packet gap for IFTP that makes both connections transmit the same number of packets in the same round trip.

Assume the following parameters:

P = Packet size in bytes.

τ = Transmission time of a single packet (in sec).

G_i = Inter-packet gap for IFTP, i.e. the time period (in sec.) between transmitting the last bit of one packet until transmitting the first bit of the next packet. The subscript i represents the adjustment (round trip) period under consideration. Thus in the initial adjustment period, the inter-packet gap is G_0 . In the next period, a gap of the length G_1 is used to separate successive packets and so on.

R = Smoothed round trip time (in sec.) between the sender and the receiver which is also equal to the length of each adjustment period for IFTP. In this paper we assume a WAN Internet environment and hence $R \gg$ TCP window.

Consider now the i^{th} period and assume that IFTP transmits n packets during this period.

Slow Start Phase:

The slow start phase is enforced by TCP as long as no packets are lost and the *ssthresh* threshold has not been reached. During slow start, TCP doubles the number of packets in the window every round trip. Therefore in the $i+1^{th}$ period IFTP should send $2n$ packets similar to TCP. Thus the number of packets that should be sent in the $i+1^{th}$ round trip period is governed by the following equation:

$$n \times (G_i + \tau) = 2n \times (G_{i+1} + \tau) = R \dots\dots\dots(1)$$

Where $n < ssthresh$. Thus in the slow start phase, the inter-packet gap in the period under consideration can be obtained using equation 2.

$$G_{i+1} = \frac{1}{2} \times (G_i - \tau) = \frac{R}{2n} - \tau \dots\dots\dots(2)$$

Next, we need to define how IFTP exits the slow start phase. This means we need to choose *ssthresh* for IFTP to be "friendly" with TCP. If *ssthresh* for IFTP is larger than that of TCP then TCP will enter congestion avoidance first, while IFTP is still doubling its bandwidth. We adopt the policy of using a common *ssthresh* for IFTP and TCP. Both IFTP and TCP can choose *ssthresh* based on the "bandwidth * delay" product [7]. In this way, we can guarantee that both IFTP and TCP have fair values of *ssthresh*.

Congestion Avoidance Phase:

In the congestion avoidance phase, IFTP mimics TCP by increasing the number of packets in each round trip linearly. That is, in the $i+1^{th}$ period, IFTP should send $n+1$ packets. Equation 3 shows this relation. From equation 3,

we can calculate the inter-packet gap that should be used during transmission in the congestion avoidance phase as shown in equation 4.

$$n \times (G_i + \tau) = (n+1) \times (G_{i+1} + \tau) = R \dots\dots(3)$$

$$G_{i+1} = \frac{n \times G_i - \tau}{n+1} = \frac{R}{n+1} - \tau \dots\dots\dots(4)$$

Where n is the number of packets successfully sent in the i^{th} period.

Packet Loss:

If a packet is lost and three packets after the missing one are received, we would like IFTP to reduce the number of packets by half to roughly emulate the behavior of TCP after fast recovery.

In this case, the number of packets sent in the $i+1^{th}$ period and the corresponding inter-packet gap can be controlled using equations 5 and 6 respectively.

$$n \times (G_i + \tau) = \frac{n}{2} \times (G_{i+1} + \tau) = R \dots\dots\dots(5)$$

$$G_{i+1} = 2G_i + \tau = \frac{2R}{n} - \tau \dots\dots\dots(6)$$

From equation 6, we notice that the inter-packet gap needs to be more than doubled to cut the number of packets by half.

4 The IFTP Protocol:

The most essential property of IFTP that makes it different than a non-IFTP such as UDP is that IFTP uses ACKs in order to be compatible with TCP. The ACK scheme for IFTP, however, does not need to be a robust one as that of TCP; a relaxed ACK approximation similar to that of the unreliable RAP protocol [5] would be sufficient for IFTP. Unlike TCP, the IFTP algorithm does not provide any reliability by retransmitting lost packets. IFTP keeps track of ACKs received and round trip times only to adjust its transmission rate in a manner that is friendly to TCP. To do so, IFTP adjusts the value of inter-packet gap (IPG) which is a controllable parameter. Thus, as stated earlier, IFTP can afford to have a crude approximation of the ACK scheme.

The Internet friendly scheme starts by setting n to 1. It sends one packet and waits for its ack to arrive to set the initial value of the round trip time (RTT). The implementation has four main parts:

1. Slow-start
2. Congestion avoidance
3. Packet Loss
4. Timeout

In the **slow start** phase, the IFTP sender doubles the number of packets to be transmitted every RTT and updates G as calculated in equation (2). The sender does

that when all the ACKs of the packets transmitted in the previous RTT are received. The sender uses the variable *ack_count* to keep track of the number of ACKs received in the current RTT. The IFTP sender exits the slow start phase when *n* reaches a threshold (*ssthresh*). It then enters the **congestion avoidance** phase where it increases *n* linearly every RTT. It updates *G* according to equation (4). This phase continues until *n* reaches the receiver's maximum advertised window. The sender uses the routine *Transmit_window* to send the required number of packets in each round trip. *Transmit_Window* is a thread that executes in parallel once it is activated from the main loop to perform one sweep for the transmission of the *n* packets of the new window.

Unlike a TCP receiver, an IFTP receiver does not use cumulative ACKs. An IFTP receiver merely transmits the sequence number of the packet it receives. If the sender detects a gap in the sequence number of the ACKs received, it assumes that a packet or more has been lost. This is an indication that the network is congested. When the number of ACKs received by the sender after the missing one reaches a certain threshold (*aftermissing_thresh*) it reacts as follows:

1. The IFTP sender halves its *ssthresh* or sets it to 2 whichever is larger to reduce the amount of data.
2. It halves the number of packets to be transmitted in the next RTT.
3. It sets the inter-packet gap according to equation (6).

The IFTP reacts only once upon loss detection. Subsequent ACKs received in the same round trip indicating further packet loss are ignored. The sender exits the packet loss phase when it receives an ACK with a sequence number greater than or equal to the last one it transmitted before the packet loss phase.

G to $R-\tau$ where τ is the transmission time of one packet. A window of size 1 is subsequently transmitted. Note that in the packet loss and timeout phases **no retransmission of packets take place**. Upon detection of a missing packet in either phase, IFTP reduces its transmission rate but transmits new packets. It does not keep track of which or how many packets have been lost. The bandwidth used by TCP and IFTP will be roughly the same. However, the bandwidth of TCP is expected to be lower than that of IFTP. This is because TCP retransmits all lost packets and waits (in some implementations) until it receives ACKs confirming that all lost packets have been received.

To calculate the timeout period, IFTP uses a mechanism similar to that used in TCP [8].

Unlike TCP, IFTP does not maintain any window nor retransmit any lost packets. In IFTP, fast recovery and timeouts translate into changing the rate of transmission by changing the inter-packet gap without changing the monotonously increasing order of the sequence number of the packet to be transmitted.

5 Simulation

In this section, we use the network topology of Figure 1. The routers have a single output queue for each attached link and use FCFS scheduling and drop-tail queuing. We use Reno TCP which is widely used on the Internet.

5.1 TCP friendliness

We use simulation to demonstrate the most important property of IFTP: TCP-friendliness. In this experiment, we used two IFTP sources and two TCP sources (no UDP sources were used) sharing the bottleneck link of bandwidth $B=1$ Mbps. We used the IFTP parameters to be the same as TCP unless otherwise stated. That is packet

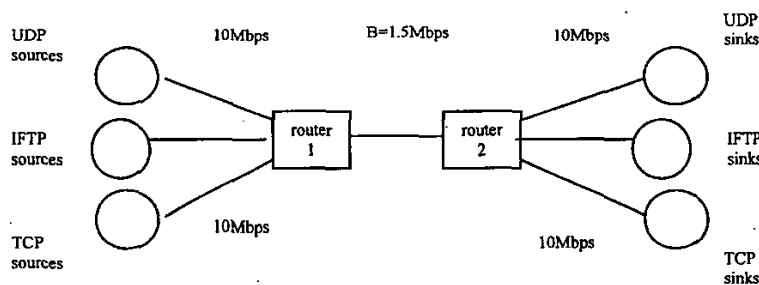


Figure 1: Simulation network

If for any reason no ACKs are received for a calculated period of time (timeout period), the IFTP sender uses its timeout mechanism. It drops *ssthresh* to half the current transmission rate. It then resets *n* to 1 and

size is equal to 1000 bytes, ACK size is equal to 40 bytes, *ssthresh* is 32, *W_max* is 50, and *dupack_thresh* is equal to *aftermissing_thresh* which is equal to 3.

We ran the simulation for 120 seconds. We calculated the average bandwidth of TCP sources and IFTP sources respectively. The results are shown in Figure 2. This

Acknowledgment

This work has been partially supported by NSF under grant 0086251 and by a grant from ARO. The

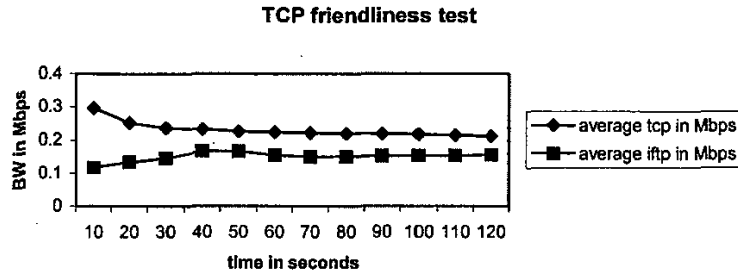


Figure 2: TCP friendliness Test

figure shows that IFTP and TCP both have a fair almost equal share of bandwidth.

5.2 Delay and Delay Jitter

We define the delay to be the time a packet is issued at the source till the time it is received. The delay jitter is the standard deviation of the delay. Table 1 shows the average delay and delay jitter for UDP and IFTP over the period of 120 seconds for the arrival rate range of UDP as above. From Table 1, we can notice that the average delay of IFTP is almost half that of UDP. Also the delay jitter of IFTP is less than that of UDP. This is because in case of UDP the packets suffer long queuing delays at the router especially as the sending rate increases.

Table 1: Delay and Delay jitter for IFTP vs. UDP

	IFTP	UDP
Delay in ms	541.05	990
Delay jitter in ms	105.42	125.45

6 Conclusion

In this paper, we presented a TCP-friendly transport protocol. IFTP is a rate based, unicast protocol. It is also an end-to-end protocol in the sense that it does not require the routers to provide any additional mechanisms to support it. IFTP is fair to good behaving protocols like TCP. If IFTP and TCP were to share a bottleneck, each flow gets a fair share of the available bandwidth. We tested IFTP's fairness and important performance metrics namely, packet delay and delay jitter. Our simulations have shown that IFTP is TCP friendly. It also provides less packet delay and delay jitter than UDP.

views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida.

References

- [1] Jacobson V. And Karel M., "Congestion Avoidance and Control", ACM SIGCOMM, pp. 314-329, August 1988.
- [2] ElAarag H. and Bassiouni M., "Performance Evaluation of TCP Connections in Ideal and Non-Ideal Network Environments", Computer Communications Journal, Elsevier Publications, v 24 n 18 pp. 1769-1779, Dec 1 2001.
- [3] Mahdavi J. and Floyd S., "TCP-Friendly Unicast Rate-Based Flow Control" Technical note sent to end2end-interest mailing list, http://www.psc.edu/networking/papers/tcp_friendly.html, January 1997.
- [4] Floyd S. and Fall K., "Promoting the Use of End-to-end Congestion Control in the Internet" IEEE/ACM Transactions on Networking, Vol. 7, No.4, pp. 458-472, August 1999.
- [5] Rejaie et al. "RAP: An End-to-end Rate-based Congestion Control Mechanism for Real time Streams in the Internet", IEEE Infocom, 1999.
- [6] ElAarag H. and Bassiouni M., "Transport Control Protocols for Wireless Connections" Proc. of IEEE Vehicular Technology Conference (VTC'99), Houston Texas, pp. 337-341, May 1999.
- [7] Hoe J. C., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", ACM SIGCOMM 1996, pp.270-280.
- [8] Fall K. and Floyd S., "Simulation based Comparisons of Tahoe, Reno, and SACK TCP", ACM Computer Communication Review, 26(3), pp.5-21, 1996.