TCP-Friendliness of Rate-based Congestion Control Protocols

Suhaidi Hassan
Faculty of Information Technology, Universiti Utara Malaysia
06010 UUM Sintok, Malaysia
{suhaidi@uum.edu.my}

***Abstract*— The main purpose of rate-based TCP-friendly congestion control protocols is to ensure that the application's traffic shares the network in a fairly and friendly manner with the dominant TCP traffic. In this work, we compare the performance of two rate-based TCP-friendly congestion control protocols, namely the *Rate Adaptation Protocol* (RAP) and *TCP-Friendly Rate Control Protocol* (TFRC). Our experimental results reveal that the equation-based TFRC is able to achieve throughput that is close to the throughput of a TCP connection using the same network path under the same network conditions. Also, the results demonstrate that the TFRC is friendlier and robust in most of our experiments, as compared to RAP.**

Keywords: Communications software, congestion control protocols

I. Introduction

Unfair competition for bandwidth among TCP and non-congestion-controlled applications (such as UDP) can cause major threats to the Internet. TCP's rapid back-offs during the congestion period make it vulnerable to bandwidth stealing by non-congestion-controlled applications. Thus, deploying a large scale of such non-congestion-controlled applications in the Internet might result in extreme unfairness towards competing TCP traffic. It is therefore important for the non-TCP applications to be designed in such a way that they are responsive to network congestion as well as being *friendly* to the competing TCP applications in terms of bandwidth sharing.

The classic example of TCP-unfriendliness is provided by UDP-based applications which do not employ congestion control mechanisms and their negative impacts are discussed in [1]. This problem appears because the UDP traffic does not respond to congestion signals, which causes TCP flows to back off. The UDP traffic continues to dominate the bandwidth, which negatively affects the throughput of the other *good* network citizen. To overcome this problem, it is crucial for such applications to employ TCP-friendly congestion control mechanisms.

The main purpose of rate-based TCP-friendly congestion control protocols that have been developed to support the deployment of multimedia applications in the Internet is to ensure that the application's traffic shares the network in a fairly and friendly manner with the dominant TCP traffic. In this paper, we compare the performance of two rate-based TCP-friendly congestion control protocols, namely the *Rate Adaptation Protocol* (RAP) and *TCP-Friendly Rate Control Protocol* (TFRC). Both TCP-friendly protocols use different rate-adaptation schemes: TFRC employs the equation-based scheme, while RAP employs the AIMD-based scheme.

The investigation on the issue of friendliness among the competing protocols is very important to enable the protocol researchers to gain a greater understanding about the resource (especially bandwidth) utilisation among competing protocols that co-exist on a network path. Friendliness behaviour among these competing protocols is crucial in safeguarding the stability of the Internet. This means that the TCP-friendly applications, while attempting to improve their bandwidth utilisation, must also be fair towards the competing TCP applications.

This paper is organised as follows. In Section II, a brief background is presented on the TCP-friendly rate-based control protocols used in the study. The experimental design and its rationale are described in Section III. In Section IV, the results of the simulation experiments are discussed, and the protocols' performance is evaluated. Finally, Section V concludes the paper.

II. Background

A. Rate Adaptation Protocol (RAP)

RAP [2] was developed by Rejaei et al. at the University of Southern California as part of an end-to-end QoS architecture. RAP is a sender-based congestion control scheme. Each data packet sent by the RAP sender is acknowledged by the receiver. The congestion detection is based on packet losses, as in TCP where the ACKs are used to detect packet loss and infer the RTT. Packet losses are determined when there are gaps in the sequence number of the transmitted packets as well as when transmission timeouts occur. Due to the fact that real-time streaming applications are essentially semi-reliable, RAP decouples congestion control and error control, leaving the application layer to deal with the latter. The receiver module observes these gaps and notifies the sender accordingly.

Using the acknowledgement from the receiver, the sender can estimate the RTT and losses. The RAP sender keeps a history of each packet sent out and not yet acknowledged or lost. This transmission history is used to estimate the RTT as well as to detect losses. The estimation of the round-trip time (RTT), called the *Smoothed* RTT (SRTT), is computed as the *exponential weighted moving average* (EWMA) of RTT samples, where $SRTT = \frac{7}{8}SRTT + \frac{1}{8}RTT_{sample}$. RTT samples can be gathered from the time interval between the sending of a data packet and the reception of the corresponding ACK. RAP computes the timeouts based on *Jacobson/Karel's* TCP algorithm [3], that $timeout = SRTT + 4SRTT_{var}$ where $SRTT_{var}$ is the variations of *SRTT*, computed as $SRTT_{var} = \frac{1}{4}|SRTT - RTT_{sample}| + \frac{3}{4}SRTT_{var}$.

The protocol uses the *Additive Increase Multiplicative Decrease* (AIMD) approach to emulate the TCP behaviour in the rate-based environment. Using AIMD, the source's sending rate is increased additively and repeatedly when there is no congestion, but when congestion occurs, it decreases the rate instantly in half. The source's sending rate is changed by reducing the interpacket gap (IPG) between the transmitted packets, which consequently increases the transmission rate. In the event of no losses, the sender depends on RTT estimation in adjusting the sending rate.

There are two types of rate adaptation employed by RAP: *coarse grain* (CG) and *fine grain* (FG) adaptation. In CG adaptation, RAP merely uses the AIMD approach for rate adjustment. The use of fine grain adaptation is intended to make RAP more stable and responsive to transient congestion. In this work, both types of RAP adaptation granularities are used for comparison against the other protocol under study.

B. TCP-Friendly Rate Control Protocol (TFRC)

TFRC [4] is a rate-based, end-to-end congestion control protocol which is intended for unicast playback of Internet streaming applications. It was developed at ACIRI by Floyd et al. Like RAP, TFRC is a source-based, rate control protocol. The sender uses the slow start technique at the beginning of the transmission phase, during which it tries to increase its sending rate multiplicatively at every RTT until it detects a loss. Packet losses are identified by gaps in the sequence number of the transmitted packet at the receiver module.

The receiver measures the packet loss rate and feeds this information back to the sender at regular intervals. The sender uses the feedback information to measure the RTT to the receiver. Like RAP, it also uses the exponential filter $t_{RTT} = \beta RTT_{sample} + (1 - \beta)t_{RTT}$ to maintain the fine time granularity, where β equals $\frac{1}{8}$, t_{RTT} is the round trip time, and RTT_{sample} is the sample of RTT gathered from the time interval between the sending of a data packet and the reception of the corresponding ACK. However, the retransmission timeout t_{RTO} is calculated as $t_{RTO} = 4t_{rtt}$ which is found to work reasonably well in providing fairness with TCP [4]. This simple empirical calculation of t_{RTO} is due to the fact that the use of the timeout value is less important in TFRC as compared to TCP. Recall that in TCP, the accurate value of t_{RTO} is needed for scheduling the packet transmissions, which is not the case with TFRC. In TFRC, this timeout value is used only for the estimation of TCP throughput.

In order to derive an acceptable TCP-friendly transmission, the TFRC sender adjusts its transmission rate based on the measured loss rate and RTT. The adjustment of the sending rate to achieve TCP-friendliness is based upon a control equation derived from the TCP throughput model. The control equation is as follows:

$$\tau_{TFRC} = \frac{s}{t_{RTT}\sqrt{\frac{2p}{3}} + t_{RTO}\sqrt{\frac{3p}{8}}p(1+32p^2)} \qquad (1)$$

where s is the maximum segment (packet) size, t_{RTT} is the round trip time, p is the probability of loss event, and t_{RTO} is the retransmission timeout.

The transmission rate of the sender is adjusted directly to match the calculated transmission rate. The rate adjustment process is made periodically at a certain interval. In the event of packet losses, the sender restricts its sending rate to the equivalent TCP rate using Equation 1. Otherwise, the transmission rate is doubled.

III. Experimental Design and Rationale

A suite of experiments is conducted in this study to investigate how the throughput of the flows is affected as a result of the rate adjustment process performed by the TCP-friendly sources when competing with their TCP counterparts in different network scenarios. In doing this, the throughputs of each rate-based and TCP flows are respectively measured, and the average bottleneck bandwidth share of each of the flows is calculated based on their average throughput. The aim is to determine the degree of friendliness (i.e. fairness) of the TCP-friendly rate-based protocols by comparing their average bandwidth share against that of the TCP flows. We used the simulator ns-2 [5] to perform our simulation experiments.

The first step to determine the degree of friendliness is to calculate the bandwidth share of the bottleneck link based on the throughput ratio of the competing connections. Simply stated, the friendliness ratio, F_r can be expressed as [6]:

$$F_r = \frac{T_F}{T_T} = \frac{\frac{\sum_{i=1}^{k_f} T_i^f}{k_f}}{\frac{\sum_{i=1}^{k_t} T_i^t}{k_t}} \qquad (2)$$

k_f denotes the total number of monitored TCP-friendly connections, and k_t denotes the total number of monitored TCP connections. Throughput of the TCP-friendly connections is denoted by $T_1^f, T_2^f, \ldots, T_{k_f}^f$, while the throughput of the TCP connections is denoted by $T_1^t, T_2^t, \ldots, T_{k_t}^t$.

As generally known, simulation experiments involving TCP normally have to deal with a wide range of variables, environments and implementation flavours (or variants). This causes difficulty in isolating a particular variable and studying its relation with a particular parameter because of interdependency among these variables. To minimise the related problems, the scope of comparison is limited to those related only to the following experiments involving:

a. *Queue management policy*. In this set of experiments, we use RED [7] queue management policy as a comparison with the normal DropTail policy. RED is known for its ability to distribute the losses evenly across the competing flows and avoid buffer overflow over a wide range of connections.

b. *Bottleneck delay.* A smaller bottleneck delay indicates a smaller round-trip time. Connections become more aggressive and achieve a larger share of bandwidth with a shorter RTT.
c. *Bottleneck bandwidth.* TCP uses the AIMD algorithm. At a smaller bottleneck bandwidth, TCP losses become more dominant, thus TCP deviates from the AIMD algorithm - paving the way for non-TCP applications to get a bigger share of the bandwidth.
d. *Loss rate.* The use of Equation 1 in controlling the sender's rate warrants that at a loss rate higher than 5%, a rate-based control protocol will still be TCP-friendly.

The evaluation of a TCP-friendly protocol behaviour includes comparing the average bandwidth share obtained by both protocols when competing with TCP, respectively under different scenarios. We compare the performance of the rate-based protocols against two TCP implementations, namely *TCP Reno* [8] and *TCP with Selective Acknowledgements (SACK)* [9]. Both are popular TCP implementations nowadays. Currently, most Internet traffic is TCP Reno-based [10], [8], while TCP SACK is currently gaining popularity and is implemented and deployed in many commercial and experimental products, such as in Microsoft Windows 98, Linux 2.1.9, and Digital Unix 4.0.

In these simulation experiments, we use two sets of competing protocols, namely the TCP friendly protocols (RAP and TFRC) and the two variants of TCP (Reno or SACK). We attach an FTP application to the TCP sources. Correspondingly, we use an application that produces a constant bit rate (CBR) traffic pattern for the TCP-friendly sources. The intermediate routers are configured using DropTail and RED queue management policies. Packet losses are simulated by packet drops at overflowed router buffers.

Figure 1 illustrates the network topology used in the exper-

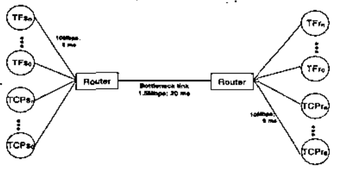

Fig. 1. Simulation topology for TCP-friendly experiments

iments. Two sets of (n+1) competing sources are used, where $0 \leq n < \infty$. One set of these sources act as the TCP-friendly rate-based source (either RAP or TFRC) transmitting TCP-friendly traffic into the network, and another set running as TCP sources with TCP Reno or SACK. The TCP-friendly source TFs$_n$ sends data to (and receives acknowledgements from) the receiver/sink agent TFr$_n$. Similarly, TCP source TCPs$_n$ sends data to (and receives acknowledgements from) TCPr$_n$ receiver agent.

A fair comparison can only be achieved with careful selection of simulation parameters. Similar parameter values are used for all flows wherever possible. For this purpose, a source packet size of 1000 bytes and ACK size of 40 bytes are used. The intermediate routers are connected by a bottleneck link with initial bandwidth set to 1.5 Mbps and initial link delay of 20 ms. The flows from both sources share the bottleneck bandwidth. The TCP flow is an FTP session and has unlimited data to send. Side links connected to the bottleneck link have bandwidth of 10 Mbps with a 6ms delay factor. The routers have a single output queue for each attached link, and initially use RED queue management policy. The second round of the simulation uses the DropTail policy as an alternative to RED. We use the standard (default) *ns-2* parameters for configuring the routers. All simulation experiments were run with a simulation length of 100 seconds, i.e. long enough for achieving their steady state behaviour.

IV. Results and Discussions

A substantial number of simulation experiments have been conducted to evaluate the performance of these two TCP-friendly protocols under various simulation scenarios. The following subsections present our results for these experiments. All the results, where appropriate, are obtained using a level of confidence of 95%.

A. Queue Management Policies: DropTail versus RED

In the first set of experiments, the performance of TCP-friendly protocols when using the DropTail queue management policy, is compared. It is discovered that in general the TCP-friendly protocols show poor friendliness results in all the experiments using DropTail as queue management. Nevertheless, TCP-friendly protocols are friendlier with TCP SACK as compared to TCP Reno. TCP SACK obtains more bandwidth share in all the cases, which therefore leads to higher throughput.

From these results, we can conclude that the use of DropTail queue management does not facilitate a fair bandwidth sharing of the TCP-friendly protocols coexisting with either TCP Reno or SACK. Thus, the use of DropTail queue management does not enhance the friendliness of TCP-friendly protocols.Unlike TCP Reno, TCP SACK senders transmit only those segments that have been lost, enabling them to obtain a bigger share of the bandwidth. This in turn affects the friendliness of the competing TFRC. However, this transmission feature of TCP SACK cannot beat the aggressiveness of CG RAP connections which, combined with the inefficiency of the DropTail queue management, are able to dominate the bottleneck bandwidth.

In the second set of experiments, we evaluate the performance of TCP-friendly protocols versus TCP Reno and SACK using RED as a queue management policy. It is observed that when using RED at intermediate routers, the TCP-friendly protocols become friendlier towards competing TCP Reno connections. The average throughput of TCP Reno connections is slightly higher than that of the TFRC. The higher throughput can result from the ability of TCP Reno to acquire more bandwidth share.

TFRC shows the best friendliness result when competing with TCP SACK. This implies that the bandwidth is fairly shared by the two competing protocols. The performance of RAP against TCP Reno is also improved when using RED, even though the results are still considerably unfavourable.

In the case of RAP versus TCP SACK, the performance of CG RAP is slightly better with RED. The performance of FG RAP improves considerably when using RED instead of DropTail. FG RAP is able to capture properly the short-term trends in congestion under heavy load. Consequently, FG RAP can respond to the distress of TCP SACK, making it friendlier in seizing bandwidth which leads to its favourable throughput.

In summary, the friendliness of TFRC is better compared to RAP, regardless of the queue management policy implemented at the intermediate routers. The use of RED improves the friendliness of RAP and TFRC. Again, TFRC scores better compared to RAP when using RED. Within TFRC experiments, TFRC works achieved better friendliness results with TCP SACK compared to TCP Reno. The sharing of bottleneck bandwidth is relatively fairer in the case of TFRC versus SACK compared to TFRC versus TCP Reno.

B. Varying Bottleneck Delay

In this experiment, we vary the bottleneck delay from 10 ms to 50 ms. For simplicity of analysis, we use a total of 30 competing TCP and TCP-friendly connections. Figure 2 shows

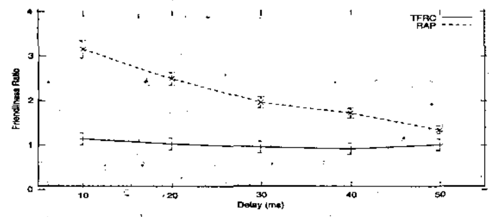

Fig. 2. Effects of varying bottleneck delay on TCP-friendliness

the simulation results.

It is observed that the friendliness of RAP improves as the bottleneck delay is increased. At a lower bottleneck delay, the RAP connections are more aggressive in obtaining the bandwidth share. As we increase the delay, the TCP SACK connections obtain more share of the bottleneck bandwidth, thus improve the friendliness results. On the other hand, the TFRC connections achieve a more stable performance against the competing TCP SACK connections as we increase the delay, compared to the RAP connections. The friendliness ratio of TFRC is not affected by the change of the bottleneck delay, indicating that both TFRC and TCP connections achieve a fair bandwidth sharing.

C. Varying Bottleneck Bandwidth

In this experiment, the bottleneck bandwidth is increased from 0.5 Mbps to 10.0 Mbps while keeping other parameters constant. Again, for simplicity of analysis, we use only 15 sources of TCP-friendly protocols competing with 15 TCP counterpart sources in this experiment. The simulation results are displayed in Figure 3.

When the bottleneck bandwidth is increased from 0.5Mbps to 3.0Mbps, the TCP-friendly connections become friendlier with the TCP connections. As the bandwidth is increased from 3.0Mbps to 10.0Mbps, the TFRC connections start to show

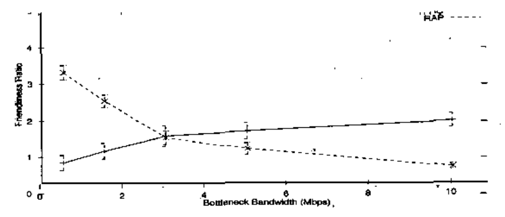

Fig. 3. Effects of varying bottleneck bandwidth on TCP-friendliness

the sign of unfriendliness towards the TCP connections. The TFRC connections receive more bandwidth share compared to the TCP connections. In contrast, the RAP connections get a decreased share of bottleneck bandwidth as the bandwidth is increased from 3.0Mbps to 10.0Mbps.

D. Loss Rate

In this set of experiments, the effect of increasing the loss rate on the TCP-friendly protocols is explored. We conduct the experiment using three different loss rates, which are 1%, 5% and 15%. The AIMD-based rate adaptation scheme will still be TCP-friendly at these loss rates up to 5% [11]. Tables I shows the simulation results.

The results clearly show that RAP, which uses AIMD to control its rate, begins to perform unsatisfactorily at loss rate increase of 5%. With the same increase, TFRC is still behaving satisfactorily in terms of friendliness with TCP. With fewer packet losses in the network links, the TCP-friendly connections behave adaptively, thus friendlier to TCP connections.

TABLE I
EFFECT OF INCREASING LOSS RATE ON RAP AND TFRC

RAP	Min. F_r	Max. F_r	TFRC	Min. F_r	Max. F_r
1%	1.21	1.36	1%	0.89	1.01
5%	1.40	1.62	5%	0.95	1.09
15%	1.98	2.33	15%	1.04	1.42

V. CONCLUSION

This work presented a performance comparison of two TCP-friendly rate-based adaptation protocols, namely RAP and TFRC, conducted using a set of experimental suites. Both TCP-friendly protocols use packet loss and RTT estimations to adjust dynamically the transmission behaviour of the sender. Our simulation results suggest that the equation-based TFRC is more TCP-friendly over the wide range of tested parameters, as compared to RAP. Also, our results suggest, in general, that the use of TCP SACK can facilitate better bandwidth sharing among the competing connections, thus enhancing throughput of the connections.

In addition, the experimental results show that both protocols perform better in the presence of RED as queue management policy. This is mainly due to the ability of RED to distribute losses evenly across the flows and avoid buffer overflow over a wide range of connections. The effects of varying the bottleneck delay and bottleneck link bandwidth on friendliness of TCP-friendly protocols have also been explored. The results also show that these protocols especially perform better at lower loss rate below 5%. At loss rates higher than 5%, bandwidth sharing with the competing TCP connections is no longer fair, resulting in poor friendliness results.

Our experimental results reveal that the equation-based TFRC is able to achieve throughput that is close to the throughput of a TCP connection using the same network path under the same network conditions. Also, the results demonstrate that the TFRC is friendlier and robust in most of our experiments, as compared to RAP.

References

[1] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transcactions on Networking*, vol. 7(4), pp. 458–472, August 1999. http://www.acm.org/dl/.

[2] R. Rejaie, M. Handely, and D. Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet," in *Proc. of IEEE INFOCOM99*, vol. 3, (New York, NY), pp. 1337–1345, March 1999. http://netweb.usc.edu/reza/pub.html.

[3] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer." RFC 2988, November 2000. http://www.ietf.org/rfc.html.

[4] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," Technical Report TR-00-003, International Computer Science Institute, March 2000.

[5] T. V. Project, "The Network Simulator - ns-2." URL: http://www.isi.edu/nsnam/ns/.

[6] S. Hassan, *Simulation-based Performance Evaluation of TCP-Friendly Protocols for Supporting Multimedia Applications in the Internet.* PhD thesis, School of Computing, University of Leeds, August 2002.

[7] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[8] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms." RFC 2001, January 1997.

[9] S. Floyd and K. Fall, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, vol. 26, pp. 5–21, July 1996.

[10] S. Bhattacharjee, *Active Networks: Architectures, Composition, and Applications.* PhD thesis, Georgia Institute of Technology, 1999.

[11] J. Mahdavi and S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control." Technical note sent to the end2end-interest mailing list, January 1997.