

A Framework For Managing Emergent Transmissions In IP Networks

Yen-Hung Hu

Robert Willis

Hyeong-Ah Choi

Department of Computer Science
Hampton University
Hampton, Virginia 23668
yenhung.hu@hamptonu.edu

Department of Computer Science
Hampton University
Hampton, Virginia 23668
robert.willis@hamptonu.edu

Department of Computer Science
The George Washington University
Washington, DC 20052
hchoi@gwu.edu

Abstract—Theories and experiments have shown that average end-to-end delay of flows increases and TCP becomes inefficient and instable during network congestion period, regardless of which queue management and packet scheduling mechanism are being applied. Despite the concerns of network congestion problem have raised by the implementations of various per-flow based transmission protocols and queue policies on routers, there are still very few efforts to assess the feasibility of modifying network infrastructure to accommodate these new algorithms. It is practically impossible to reach the performance that claimed by the per-flow based algorithms, while updating routers is very costly and beyond the control of every individual network domain. In this paper, we present an application layer reliable multi-connection best effort framework which provides quality of service (QoS) for emergent communications with no or very few changes of the existing network infrastructure. Performance of our approach and other several existing algorithms is studied by using the network simulator NS2.

I. INTRODUCTION

In the current IP networks, most of the existing congestion control technologies are based on queue management and packet scheduling policy at each router. The first-in-first-out (FIFO) scheduling with drop tail (DT)) queueing is the simplest queue management policy for congestion control applied in most routers. The random early detection (RED) [1] is the most popular active queue management scheme in which the drop policy is dynamically changing in response to network traffic conditions. These schemes work well when their traffics are properly implemented TCP flows. Several alternatives have been proposed to improve the situation with the objective of allocating a fair share of bandwidth to each flow. They include SRED [2], FRED [3], deficit round robin (DRR) [4], and fair queue algorithms [5].

Despite the performance improvement that per-flow based queue management algorithms suggested in the literatures has been achieved in experimental environments, there are significant drawbacks when they are implemented in real networks. Every router traversed by active flows must adopt these queue management algorithms in advance. However, algorithms are usually implemented by router manufactures and selected based on the Internet infrastructure, and updating all routers is very costly and beyond the control of every

individual network domain. Therefore, adopting new queue management algorithms by modifying the current Internet infrastructure is practically difficult.

When networks are congested by the increasing non-responsive¹ traffic, TCP becomes inefficient and instable, regardless of which queue management and packet scheduling mechanisms are being applied. It was reported that even the per-flow based queue algorithms reveal a significant shortcoming in protecting packets from normal flows when large amount of non-responsive flows exist [6], [7]. Therefore, using TCP protocol and existing queue management algorithms to transmit emergent messages (i.e., strict QoS demand services, such as military communications, emergent public service reports and resource reports, etc.) during network congestion period is not an appropriate approach.

In this paper, we propose a framework for managing emergent transmissions with no or very few changes to the existing Internet infrastructure. In our previous researches [8], we have observed: (1) UDP with proper packet lost indication and retransmission mechanisms could provide better performance on QoS than TCP when a network is congested by high rate malicious flows. (2) a flow with smaller size will have better survivability and end-to-end delay than a flow with larger size. These two observations give us the ideas of designing our application layer reliable multi-connection best effort framework.

The remainder of the paper is organized as follows. In Section 2, we describe theoretical analyses of the end-to-end delay of UDP and TCP flows. Section 3, we show simulation results of larger and smaller flows while they are affected by a high rate malicious flow. Section 4, we discuss the variation of end-to-end delay while a single connection transaction is planing to transmit the same amount of data as a multiple connections transaction. Section 5, we depict our framework with rationale analyses. Section 6 conclude the paper.

¹Non-responsive means that it will not reduce sending rate while experiencing packet losses.

II. END-TO-END DELAY ANALYSIS OF TCP AND UDP

In this section we study stochastic models for congestion control that yield relatively simple analytic expressions for the end-to-end delay² of TCP and UDP.

A. Model of TCP

TCP is a responsive protocol which reacts to network congestion through complex mechanisms including *slow start*, *congestion avoidance*, and *fast retransmission*. At the beginning, TCP starts its congestion window (cwnd) at 2 and sends out 2 packets (the same number as the congestion window size) at once. TCP increases its congestion window size by 1 each time when a non-duplicate acknowledgment (ACK) is received. However, this increment will be tuned to $1/cwnd$ when the current congestion window size is larger than the slow start threshold (ssthresh) (This is called congestion avoidance). Generally, TCP uses two mechanisms to evaluate network congestion (i.e., packet losses): multiple duplicate acknowledgments (dupacks) and retransmission time out (RTO). If the notification of packet losses is due to retransmission time out, TCP will reset the congestion window to the initial size (i.e., 2) (this is called slow start). However, if the notification of packet losses is due to the reception of multiple duplicate acknowledgments, TCP will shrink the congestion window to half of the previous size (this is called fast retransmission). In both notification cases, the slow start threshold will reduce to half of the congestion window size.

In this paper, we focus our attention on the congestion avoidance behavior of TCP and the impact on throughput and end-to-end delay which are using as the assessment of QoS.

We consider a TCP flow i starting at time $t_i = 0$, where the sender always has data to send until all packets are transmitted. We assume flow i has N_i packets, and needs t_i ($[0, t_i]$) seconds to transmit them. Let B_i be the throughput (i.e., sending rate) of flow i . We have

$$B_i = \frac{N_i}{t_i} \quad (1)$$

Therefore, for the receiver of TCP flow i , it needs at least $t_i + \kappa_i RTT_i$ seconds to receive all N_i packets (see Figure 1). Where, RTT_i is the average round trip time of flow i . κ_i is a constant which relates to packet loss of the last round. If there is no packet loss during the last round and no packet retransmission involved, then κ_i will be 1. However, if there are packet losses during the last round, then κ_i will be larger than 1 and depends on how long it needs to retransmit all lost packets.

Therefore, the delay D_i of TCP flow i will be

$$D_i = t_i + \kappa_i RTT_i = \frac{N_i}{B_i} + \kappa_i RTT_i \quad (2)$$

²Because of retransmitting all lost packets, in TCP, end-to-end delay means the period between the time stamp of the first packet being sent and the time stamp of the last packet being received. But in UDP, packets may lose without being retransmitted, end-to-end delay may only reflect the period between the time stamp of the first packet being sent and the time stamp of the last successfully received packet.

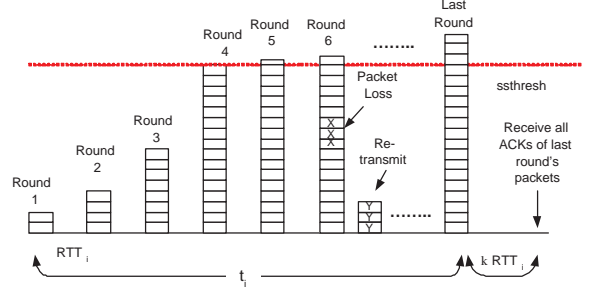


Fig. 1. Delay of TCP flow i . If there is no packet loss in the last round, $\kappa_i = 1$ and delay $D_i = t_i + RTT_i$. Otherwise, $D_i = t_i + \kappa_i RTT_i$. Where, κ_i depends on the time needed to retransmit all lost packets of the last round, X indicates a lost packet, and Y indicates a retransmitted packet.

if $N_i/B_i \gg \kappa_i RTT_i$, then $D_i \approx N_i/B_i$.

Let j be a router passing through by TCP flow i , q_j is its average queue size, and C_j is its capacity. Therefore average round trip time of TCP flow i could be demonstrated as

$$RTT_i \approx a_i + \sum_j \frac{q_j}{C_j} \leq a_i + \sum_j \frac{q_j^{max}}{C_j} \quad (3)$$

where a_i is the fixed propagation delay, q_j/C_j models the queuing delay of router j , and q_j^{max} is the maximum queue size of router j .

To discuss the throughput of TCP flow i , we adopt the result from [9] and rewrite B_i as

$$B_i(p_i) \approx \frac{1}{RTT_i \sqrt{\frac{2p_i}{3}} + T_0 \min(1, 3\sqrt{\frac{3p_i}{8}}) p_i (1 + 32p_i^2)} \quad (4)$$

where T_0 is the retransmission time out, and p_i is the packet loss rate of TCP flow i .

From equation 4 we have observed that the throughput B_i of TCP flow i reduces when the packet loss rate p_i increases. Therefore, the end-to-end delay for delivering all N_i packets enlarges.

B. Model of UDP

UDP is a non-responsive flow which adopts best effort policy and does not reduce the sending rate when experiencing network congestion. Meanwhile, unlike TCP, UDP does not retransmit lost packets. Therefore, the throughput of a UDP flow will keep at whatever it is at the beginning until all packets are transmitted.

Assume an UDP flow k , having N_k packets, starts at time $t_k = 0$, where the sender always has data to send until all N_k packets are transmitted. Thus the time needed for sending all packets is

$$t_k = \frac{N_k}{B_k} \quad (5)$$

Let's ignore the lost packets (Actually, UDP packet loss rate will affect quality of service, for instance, VoIP. But, this will not be discussed here.), and the end-to-end delay D_K of UDP

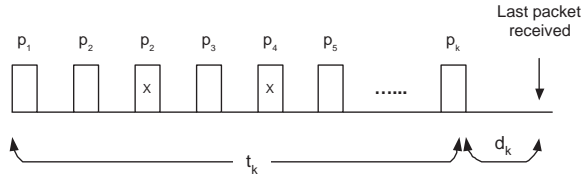


Fig. 2. Delay of UDP flow k . Delay $D_k = \frac{N_k}{B_k} + d_k$. Where, d_k depends on network propagation delay and queuing delay, and X indicates a lost packet.

flow k will be measured as the time period needed for all N_k packets to reach the destination (see Figure 2). Therefore end-to-end delay D_k of flow k can be illustrated as

$$D_k = \frac{N_k}{B_k} + d_k \quad (6)$$

where d_k is affected by network propagation delay and queuing delay. If $N_k/B_k \gg d_k$, then $D_k \approx N_k/B_k$.

C. Effect of Flow Size

Let flow i and j be two TCP flows with size N_i and N_j packets respectively. To compare end-to-end delay of these two flows, both of them are induced into the same network environments. Where, these two flows will have approximately the same drop probability, propagation delay, and queuing delay. Therefore,

$$\frac{D_i}{D_j} = \frac{\frac{N_i}{B_i} + \kappa_i RTT_i}{\frac{N_j}{B_j} + \kappa_j RTT_j} \quad (7)$$

According to Equation (4), $B_i \approx B_j$, if they have approximately the same drop probability, propagation delay, and queuing delay. Meanwhile, $D_i/D_j \approx N_i/N_j$, if $N_i/B_i \gg \kappa_i RTT_i$ and $N_j/B_j \gg \kappa_j RTT_j$.

Intuitively, we could also deduce that the end-to-end delay of a UDP flow is proportional to its throughput when network conditions are fixed (see Equation 6).

III. END-TO-END DELAY EVALUATION OF VARIOUS FLOW SIZES

In order to evaluate the effect of flow size, in this section, we study the end-to-end delay of flows, including TCP and UDP, with different flow size.

To mimic the malicious attacks, a high rate malicious flow is induced into the network. The details of the network topology is showing in Figure 3. In the network, bandwidth and propagation delay of the bottleneck ($X_1 - X_2$) are 1.5Mbps and 20 ms respectively. Bandwidth and propagation delay between the senders (include an attacker and several normal flows) and router 1 are 10 Mbps and 2 ms respectively. Bandwidth and propagation delay between receivers and router 2 are 10 Mbps and 4 ms respectively.

Evaluations are performed on NS2 network simulator. Every simulation starts at 0 seconds and end at 100 seconds. On every link, Drop Tail (DT) and first in first out (FIFO) are selected as the queue management and scheduling policy respectively.

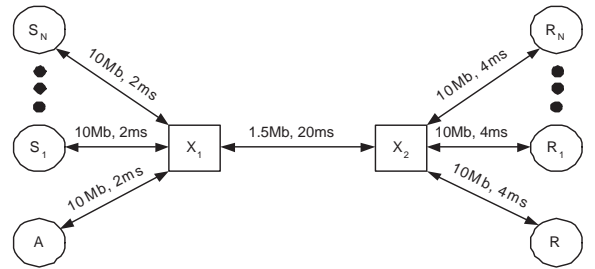


Fig. 3. Network Topology. Where, A is an attacker, R is a receiver for this attacker, X_1 and X_2 are routers, S_i is normal flows, and R_i is a receiver for S_i . The sending rate of the attacker is very high and almost saturate the bottleneck.

Buffer size is 100 packets for bottleneck link, and 50 packets for other links.

A. High Rate Malicious Flow

In our simulations, packets from the high-rate malicious flow are introduced into the network during whole simulation period. On the Internet, malicious packets may come from multiple sources and destined to multiple destinations. In this paper, as so many researchers have shown that the Internet traffic is not a traditional Poisson process but self similarity process [10]–[13] and only a few characteristics of the malicious flow are clear and available in the literature, we use the Hurst parameter (H) and rate of the aggregate traffic as the input to generate the simulated malicious traffic. We do not differentiate identifications (IDs) from every packets of the malicious flow, and we simply consider the H value and rate of the malicious flow.

A code published in <http://ita.ee.lbl.gov/html/contrib> was implemented in the NS2 simulator. This program takes the H value as an input and generate an output as a time series whose H value matches with the given input.

In this paper, we pick $H = 0.95$ (high self similarity) to indicate the major characteristic of the high rate malicious flow. When the malicious flow is the only traffic in the network, it shows bursty behaviors and almost consumes all the bottleneck bandwidth: $\text{goodput}^3 \approx 1.5 \text{ Mbps}$ (see Figure 4). Meanwhile, the high rate malicious flow is always on the network and coexists with the normal flows.

B. Various TCP Flow Sizes

To evaluate the size effect of a TCP flow while it shares the same bandwidth with a high rate malicious flow, we induce several TCP flows, having different flow size, one by one into the network which is nearly saturated by the malicious flow. Since TCP is a reliable protocol and will retransmit all lost packets, the simulation results show that the end-to-end delay of TCP (period between the time stamp of the first packet of the TCP flow being sent and the time stamp of the last packet of the TCP flow being received) is propositional to the flow size (see Figure 5). Such results support our observations of previous Section.

³Goodput means the rate of packets successfully reach the destination.

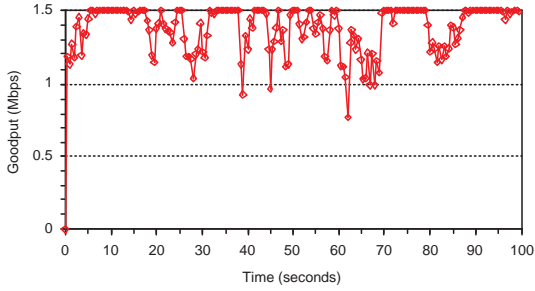


Fig. 4. Instantaneous goodput (Mbps) of the malicious flow. Where it is the only traffic and its Hurst Parameter H is 0.95.

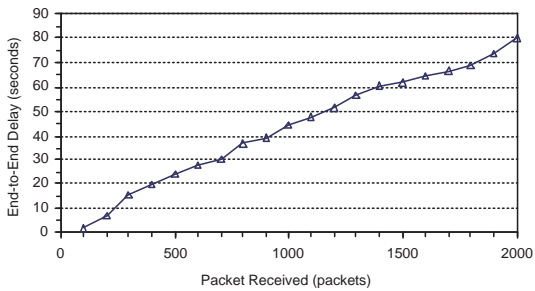


Fig. 5. End-to-end delay of TCP flows with various flow size. Slope of the data is 0.03855.

C. Various UDP Flow Sizes

We also evaluate the size effect of a UDP flow while it shares the same bandwidth with a high rate malicious flow. Since UDP adopts same congestion control policy (best effort) as the malicious flow, the end-to-end delay of UDP is also proportional to the flow size (see Figure 6) (Since UDP does not provide retransmission for the lost packets, some packets may lose during transmissions and the last received packet of the UDP flow may not reflect the last packet of the UDP flow). Such results also support our observations of previous Section.

IV. END-TO-END DELAY AND GOODPUT EVALUATIONS OF VARIOUS FLOW NUMBERS

In Section III, we have studied the effect of the flow size in terms of end-to-end delay. We have observed that: A smaller flow (both UDP and TCP) will have smaller end-to-end delay than that of the larger flow when both of them are under the same network environments. Therefore, partitioning a larger flow into several smaller flows may improve the end-to-end delay and meet the requirements of several QoS concern applications (e.g., VoIP, Audio, Video, ect.) with no modification of the existing network infrastructure.

In this Section, we study and evaluate the end-to-end delay and goodput of a larger flow (TCP or UDP) and a composite flow which is made from several smaller flows (TCP or UDP). Where, this larger flow has the same amount of packets as the composite flow. For instance, if a larger flow has N packets,

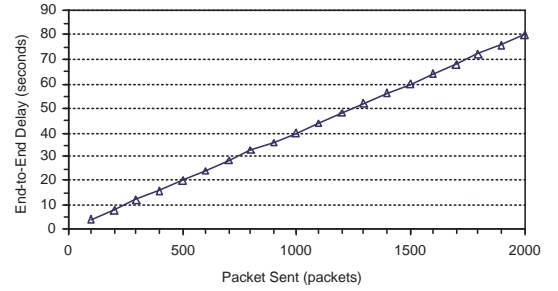


Fig. 6. End-to-end delay of UDP flows with various flow size. Where, Y-axis indicates receiving time of the last received packet of the UDP flow. Since packets may lose during transmissions, Y-axis may not reflect the receiving time of the last packet of the UDP flow. UDP's sending rate is 0.2 Mbps, and slope of the data is 0.04.

we could evenly divide it into M smaller flows, and each smaller flow will have N/M packets, and the composite flow is made by these M smaller flows. In TCP, the larger flow and the smaller flows will have same protocol setup (ssthresh, cwnd, maximum cwnd, etc.). In UDP, the large flow and the smaller flows will have same sending rate.

In our simulations, the larger flow is evenly divided into 10 smaller flows. In TCP, the larger one and the smaller ones have the same protocol setup, but the larger one will transmit 2000 packets but each of the smaller one will transmit 200 packets only. In UDP, the sending rate of all of them is 0.2 Mbps. The larger flow will send out 2000 packets but each of the smaller one will send out 200 packets only. Again, since TCP is a reliable protocol, it will retransmit all lost packets until all packets are successfully reached the destination. Since UDP is not a reliable protocol, it only sends out packets in best effort manner without any consideration of lost packets. Therefore end-to-end delay of TCP flow is counted as the period between the time stamp of the first packet of the TCP flow being sent and the time stamp of the last packet of the TCP flow being received. However, end-to-end delay of UDP flow is counted as the period between the time stamp of the first packet of the UDP flow being sent and the time stamp of the last received packet of the UDP flow (it may not be the last packet of the UDP flow).

A. Comparison of 1 Larger TCP Flow and 10 Smaller TCP Flow

First, we evaluate the end-to-end delay and goodput of the larger TCP flow and the 10-smaller-TCP composite flow. Figure 7 shows that the 10-smaller-TCP composite flow has shorter end-to-end delay than the larger TCP flows no matter what flow size they have. Meanwhile, Figure 8 shows that the 10-smaller-TCP composite flow has better goodput than the larger one. Note that, the goodput of the 10-smaller-TCP composite flow drops close to 0 Mbps after 50 seconds because most of them have finished the transmissions.

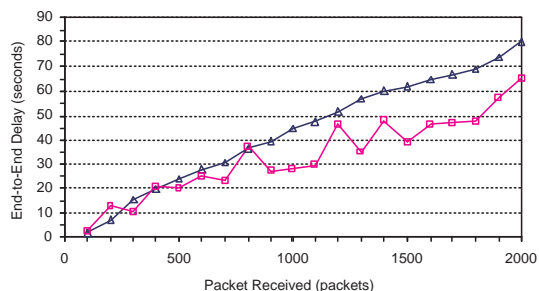


Fig. 7. End-to-End delay of one larger TCP flow (blue triangle) and one 10-smaller-TCP composite flow (pink square). In the composite flow, the delay is indicated as the largest delay among these 10 smaller TCP flows.

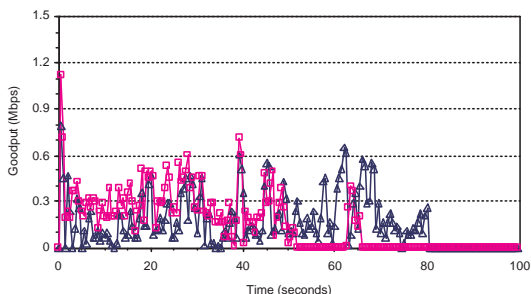


Fig. 8. Goodput of one larger TCP flow (blue triangle) and one 10-smaller-TCP composite flow (pink square). Where the receiver of the larger TCP flow receives 2000 packets, and each of the ten receivers of the 10-smaller-TCP composite flow receives 200 packets (overall is 2000 packets).

B. Comparison of 1 Larger UDP Flow and 10 Smaller UDP Flow

Now, we are examining on the end-to-end delay and goodput of the larger UDP flow and the 10-smaller-UDP composite flow. Our simulation results show that: end-to-end delay of the 10-smaller-UDP composite flow is much better than the larger UDP flow (see Figure 9), and goodput achievement of the 10-smaller-UDP composite flow is not only better than the larger UDP flow but also beats the high rate malicious flow (Initially, the malicious flow almost saturates the bottleneck, ≈ 1.5 Mbps. But, its goodput is down to about 0.4 Mbps now.) (see Figure 10). Note that, the goodput of the 10-smaller-UDP composite flow drops close to 0 Mbps after 10 seconds because most of them have finished the transmissions.

V. APPLICATION LAYER RELIABLE MULTI-CONNECTION BEST EFFORT FRAMEWORK

From the studies we have done in previous Sections, we have observed: A smaller flow (TCP or UDP) will have shorter end-to-end delay than that of the larger one when they are in a congested network; and UDP will provide better goodput achievements than TCP when they are in the same network environments.

Therefore, we have proposed a *Application Layer Reliable Multi-Connection Best Effort Framework* (see Figure 11)

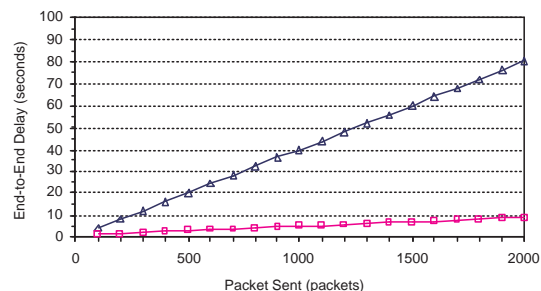


Fig. 9. End-to-End delay of one larger UDP flow (blue triangle) and one 10-smaller-UDP composite flow (pink square). Where the sending out of all UDP flow is 0.2 Mbps. In the composite flow, the delay is indicated as the largest delay of the last received packets among these 10 UDP flows.

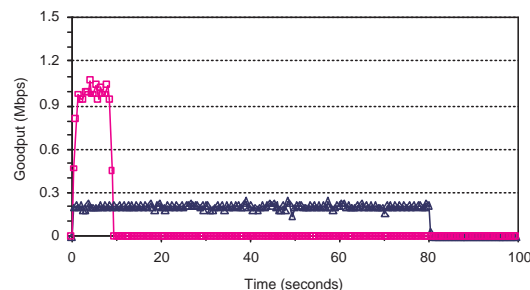


Fig. 10. Goodput of one larger UDP flow (blue triangle) and one 10-smaller-UDP composite flow (pink square). Where the larger UDP flow sends out 2000 packets, and each of the ten UDP sender of the 10-smaller-TCP composite flow sends out 200 packets (overall is 2000 packets). The sending rate of all UDP flows is 0.2 Mbps.

which follows the ideas of:

- (1) A partitioning mechanism for evenly dividing the larger flow into several smaller flows is needed because smaller flows will have better end-to-end delay than the larger one.
- (2) UDP with proper packet loss indication and retransmission mechanisms could provide better performance than TCP when network is congested by a high rate malicious flow.

A. Algorithm

In our algorithm, a specific application must be installed in both ends to provide QoS for their communications. This application uses UDP in the transmission layer and covers following 4 major functions (see Figure 12 for the details):

1. partitioning Larger Flow The larger flow f_i with N_i packets will be evenly partitioned into M_i smaller flows by using function $Partition(B_i, N_i)$. Where, B_i is the bottleneck bandwidth on the path of flow i . This function will find the optimized number of the small flows according to the bottleneck bandwidth and the large flow size.

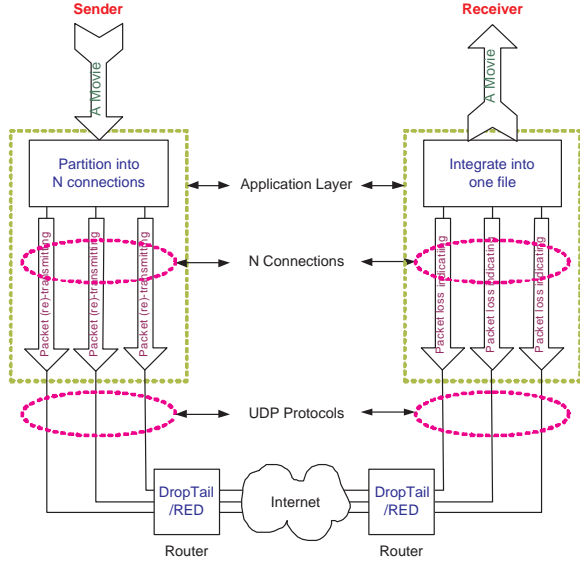


Fig. 11. Application Layer Reliable Multi-Connection Best Effort Framework.

2. Updating of Bottleneck Bandwidth: The bottleneck bandwidth that flow i (f_i) traversed will be updated every α seconds by using function $Bandwidth(f_i)$. In order to reduce overhead, in our implementation, $\alpha \geq \gamma$.
3. Retransmitting Lost Packets: The lost packets of flow i will be retransmitted every β seconds by using function $Retransmit(Lost_Buffer_i, \nu)$. Where, $Lost_Buffer_i$ stores lost packets of flow i at the sender, and ν is the number of lost packets which will be retransmitted every time. To avoid involving another burst of packet losses, retransmission will occur β seconds after detecting packet loss. Where, $\gamma \leq \gamma + \beta \leq \alpha \leq 2\gamma$.
4. Indicating Lost Packets: The receiver will indicate the sender the information of lost packets every γ seconds by using function $Lost_Indicator(loss_packet_i)$. Where, $loss_packet_i$ stores lost packets of flow i at the receiver.

VI. CONCLUSIONS

Since we have observed that: (1) UDP with the retransmission mechanisms for the lost packets could provide better performance than TCP; (2) The smaller flow will provide better end-to-end delay and goodput than the larger flow; and (3) Modification of the current Internet infrastructure to accommodate per-flow queuing algorithms may have practical difficulties. Therefore, we propose an application layer reliable multi-connection best effort framework which could satisfy the QoS requirements of several existing applications. Our algorithm has several functions for handling flow partitioning, packet lost indication, retransmission, and bandwidth exploitation. Since our framework uses multiple connections for a large flow, it can provide better survivability and end-to-end delay for emergent transmissions during network congestion period.

Algorithm 1: Application Layer Reliable Multi-Connection Best Effort Framework

f_i : flow i .
 B_i : bandwidth of the bottleneck on the path of flow i .
 $Bandwidth(f_i)$: return the value of B_i .
 f_i^j : j^{th} small flow of i .
 N_i : size of flow i .
 M_i : flow i is evenly divided into M_i smaller flows.
 $Partition(B_i, N_i)$: return the value of M_i .
 α : the period of updating B .
 β : the period of retransmitting lost packets.
 γ : the period of indicating the packet loss information to the sender.
 ν : number of packets being retransmitted.
 $lost_packet_i$: store lost packets at the receiver.
 $Lost_Indicator(lost_packet_i)$: store lost packets' ID and send it to the sender every γ seconds.
 $Lost_Buffer_i$: store lost packets at the sender.
 R_i : overall sending rate of flow i .
 $r_i = R_i/M_i$.
 $Send(r_i)$: send packets for each smaller flow j .
 $Retransmit(Lost_Buffer, \nu)$: retransmitted lost packets.

BEGIN:

1. Update bottleneck bandwidth B_i every α seconds.
 $B_i = Bandwidth(f_i)$;
2. Retransmit lost packets every β seconds.
 $Retransmit(Lost_Buffer_i, \nu)$;
3. Indicate packet loss information every γ seconds.
 - a. $Lost_Indicator(lost_packet_i)$;
 - b. add $loss_packet_i$ into $Lost_Buffer_i$;
4. At time t :
 - a. Partition flow i into M_i small flows.
 $M_i(t) = Partition(B_i(t), N_i)$;
 - b. Send packets of each sub-flow in rate $r_i(t)$.
 \forall sub-flow j of flow i do
 $Send(r_i(t))$;
 - c. If a packet p_k is lost, add it into $lost_packet_i$.
 $lost_packet_i = lost_packet_i + p_k$;
5. $t = t + \delta t$.
6. GOTO 1 until all packets of flow i are received.

END:

Fig. 12. Application Layer Reliable Multi-Connection Best Effort Framework

The performance study of our approach and other several existing algorithms is performed through the network simulator NS2. We would see that with proper tuning of the parameters, the QoS provision of emergent transmissions could be achieved with no or very few changes of the existing network infrastructure. These results encourage us to develop other application layer approaches and could embed them into the existing multi-media applications.

ACKNOWLEDGMENT

This work has been supported in part by Hampton University Research Fund (Account index 211146, Fund 100000, Org number 3372, Program 4212).

REFERENCES

- [1] Floyd, S., and Jacobson, V., *Random early detection gateways for congestion avoidance*, IEEE/ACM Trans. on Networking, 1(4), Aug. 1993.
- [2] Ott, T. J., Lakshman, T. V., and Wong, L. H., *SRED: stabilized RED*, in Proc. of IEEE INFOCOM99, New York, USA, March 1999.
- [3] Lin, D. and Morris, R., *Dynamics of random early detection*, in Proc. of SIGCOMM 97.
- [4] Shreedhar, M. and Verghese, G., *Efficient fair queueing using deficit round robin*, in Proc. of SIGCOMM 95, Cambridge, MA, USA.
- [5] Demers, A., Keshav, S., and Shenkar, S., *Analysis and simulation of a fair queueing algorithm*, J. Internetwork. Res. Experience ,pp. 3-26, Oct. 1990.
- [6] Hu, Y., Choi H., Choi, H., *Packet Filtering for Congestion Control under DoS Attacks*, in proceeding of IWIA 2004.
- [7] Hu, Y., Choi H., Choi, H., *Packet Filtering to Denfend Flooding-Based DoS Attacks*, in proceeding of IEEE Sarnoff 2004.
- [8] Hu, Y., Yu, M., Tang, D., and Choi, H., *A Study of Traffic Survivability Under Malicious Attacks*, in proceeding of IEEE Sarnoff 2006.
- [9] Padhye, J., Firoiu V., Towsley D., and Kurose J., *Modeling TCP Throughput: A Simple Model and its Empirical Validation*, in proceeding of SIGCOMM 1998.
- [10] Leland, W., Taqqu, M. , Willinger, W., and Wilson, D., *On the Self-Similar Nature of Ethernet Traffic (Extended Version)*, IEEE/ACM Transactions on Networking, Vol. 2, No. 1, pp. 1-15, February 1994.
- [11] Paxson, V., and Floyd, S., *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, Vol. 3 No. 3, pp. 226-244, June 1995.
- [12] Crovella, M. E., and Bestavros, A., *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, IEEE/ACM Transactions on Networking, 5(6):835846, December 1997.
- [13] Erramilli, A., Narayan, O., and Willinger, W., *Experimental Queueing Analysis with Long-Range Dependent Packet Traffic*, IEEE/ACM Transactions on Networking, Vol. 4, No. 2, pp. 209-223, April 1996.