Router Assisted Schemes For Performance Improvement In The Internet

Murali K. Viswanathan Ahmed E. Kamal*
Networking Research Lab
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
{muralikv,kamal}@iastate.edu

Abstract

The exponential increase in the use of Internet had necessitated a central role for congestion control. The congestion handling mechanism of the most popular protocol suite of the Internet, the TCP/IP suite, is through implicit feedback about the network conditions to the traffic source. If the feedback can be obtained sooner at the source, the congestion handling can be more effective. The goal of this work is to expedite the feedback process and enhance the performance of the Internet under congestion. We motivate the need for the proposed schemes with the help of a simplified analytical model which evaluates the time to detect a packet loss through duplicate acknowledgements. We show that this time can be significant when the number of flows going through a router is large. We then present two schemes, *Triple Packet Buffering on Congestion*, and *Triple Truncated Packets*, that enables congestion information to be fed back to the source faster. This results in a faster response to the congestion conditions, and a faster retransmission of of discarded packets. With the help of simulation results, we show that the proposed schemes perform better than the standard implementation in a variety of network scenarios and achieve higher throughput and lower mean delay. This scheme also does not require any changes in the standard TCP protocols, but a simple (optional) modification at the routers.

I Introduction

The Internet is a meta network consisting of multiple interconnected networks, supporting multiple applications and varying degrees of quality of service and security. All Internet applications use the TCP/IP suite of protocols [11]. This protocol provides end-to-end functionality at the transport layer, e.g., using the connection-oriented Transport Control Protocol (TCP), or the connectionless User Datagram Protocol (UDP). The Internet Protocol (IP) also provides the routing functionality at the network layer.

Although Internet link speeds have incrementally increased over the years, the recent explosive growth in the users size and demands has resulted in increasing cases of network congestion. Since congestion can lead to packet losses, and delay in detecting and recovering from such losses, as well as repeated consumption of network resources, congestion can adversely affect the performance of the Internet. It is therefore of paramount importance to prevent network congestion, if possible, or to detect cases of congestion as soon as they develop, and to start early recovery procedures.

Network congestion detection is implemented in the Internet by the TCP layer, and on an end-to-end basis. The TCP protocol in part tries to avoid congestion, and in another part responds to congestion by reducing the source's transmission rate. As far as this congestion control function is concerned, packet discarding due to congestion are detected through the absence of packet acknowledgements, and are used to initiate the recovery procedure. There are several versions of the TCP protocol, most importantly, Tahoe and Reno [12]. These versions differ in how they detect packet losses, and how they react to such situations.

This paper presents an approach to achieve the objective stated above, namely, two methods for expediting the detection of congestion, and the early initiation of the loss recovery procedure. Our approach

*Corresponding author; e-mail: kamal@iastate.edu; telephone: (515) 294-3580; fax: (515) 294-8432.

does not require any changes in the end-to-end or network protocols. It just requires that the network routers implement simple add-on procedures, which can be considered an option to be turned on or off by the network operator.

The paper is organized as follows. In Section II we provide a brief description of the different mechanisms implemented in the different versions of the TCP protocol. Section III.1 presents the motivation for our proposed schemes. We also use a simple analytical model to further justify this motivation. The two schemes are presented in Section III.3 and Section III.4, respectively. The performance results of these schemes are presented in Section IV, and the paper is finally concluded with a few remarks in Section V.

II Background

The TCP protocol is an end-to-end protocol, in the sense that the protocol data is only handled by the sender and the receiver, and not by the network [1]. Every segment that is transmitted by the source is kept in a buffer at the source until it is finally acknowledged by the receiver, or a timer expires, at which time it must be retransmitted. The receiver, upon receiving a data segment, whether it is the expected segment or not, always sends back an acknowledgement to the source indicating the data sequence number it is expecting. The TCP congestion control mechanism uses the sliding window mechanism, and is part preventive, and part reactive. In the preventive phase, there are two stages. In the first, called *slow start*, the source starts with a window size of one data segment, and for every acknowledgement it increases the window size by one data segment, until the window reaches a threshold level, which is initially set arbitrarily high. At this point in time, the TCP protocol enters the *congestion avoidance stage* in which the window size is incremented by one maximum segment size every time a window full of segments is transmitted and acknowledged (linear window size increase). The reactive phase takes place when a packet is lost inside the network and is not acknowledged, which causes the source to time out, reduces its window size to one maximum segment size, and sets the window threshold to [14]

$$\max(\text{unacknowledged data}/2, 2 * \text{max. segment size}) \tag{1}$$

It then enters the slow start stage again, until the window size reaches the threshold, at which time it switches to the congestion avoidance phase, and so on.

The TCP Reno version implements two other mechanisms which detect, and react to congestion faster. The first mechanism, called *fast retransmit* detects the loss of a data segment when it receives an acknowledgement followed by three duplicate acknowledgements of the same segment. The lost segment is then retransmitted. The second mechanism, *fast recovery*, follows. The source then halves the window size, sets the threshold to that new window size, increments the window size by three, enters the congestion avoidance stage, and keeps on transmitting new segments. Duplicate acknowledgements keep on incrementing the window size, while a new acknowledgement resets the window to the threshold value. The last event causes the start of the congestion avoidance phase. Therefore, the TCP Reno protocol does not implement the slow start phase, except at the start of a new connection.

Several other enhancements have been introduced to the TCP protocol over the years. For example, TCP Selective Acknowledgement (SACK) ([8]) uses the TCP header's *Options* field to indicate to the sender the non-contiguous blocks of data that have been received by the destination, which enables the source to determine the packets that have already been received and can avoid unnecessary retransmissions. Forward Acknowledgement,FACK, ([9]) uses the property of *SACK*, where the *forward-most* data at the receiver is conveyed to the sender. This provides the sender with additional information which is used to compute the actual data outstanding in the network, and transmit more data accordingly. The extension of SACK to support duplicate acknowledgments, named D-SACK, suggests that when duplicate packets are received, the first block of the SACK option field can be used to report the sequence numbers of the packet that triggered the acknowledgment. The NewReno scheme is aimed at TCP implementations that do not include the SACK option [3]. NewReno modifies the Fast Recovery algorithm to take into effect Partial Acknowledgements, wherein the new ACK conveys the receipt of a few of packets that were transmitted. [2] provides a comprehensive summary of the recent work for enhancing the TCP congestion control. References [4], [13] and [6] present more schemes for enhancing the performance of the TCP protocol.

III Router Assisted Congestion Recovery

III.1 Motivation

One of the important factors that contributes to better handling of congestion in networks is the ability to detect congestion in networks at an early stage. In TCP/IP networks, a packet loss is regarded as an indication of congestion. As discussed earlier, TCP detects packet losses by the expiry of a round trip timer timeout or through the receipt of three duplicate acknowledgments [12]. Measurements in [10] reveal that the timeouts constitute the majority or a significant fraction of the total number of loss indications. The system goes into the slow start mode if the packet loss is detected through timeouts. On the other hand, triple duplicates cause the system to go into fast retransmit and fast recovery, thus avoiding the slow start phase. This effectively means that, when there is a timeout, the system employs a congestion window size of one segment, which increases in response to received acknowledgments. Nonewithstanding the fact that this helps reduce congestion in the network, the abrupt reduction in the flow rate may lead to bandwidth wastage.

Motivated by the above, it is desirable for TCP to receive the notification of congestion through triple duplicates rather than timeouts. Furthermore, it is also desirable to detect congestion as soon as possible, and not depend on the traffic characteristics for the three duplicate acknowledgements to occur. If a particular flow is lightly loaded, once a packet is lost from that flow, it will take a long time to get three packets generated by that flow. Also, under heavy load, more packets may be lost, and it will also take a long time for three more packets to be accepted. The schemes proposed in this chapter were motivated by this observation. To further verify this intuition, we present an analytical model that approximates the behavior of the router buffer's packet handling mechanism. This model will be used to obtain an estimate for the time needed to accept three packets at a router after a packet is discarded, and thus generate three duplicate acknowledgements.

III.2 An Analytical Model

With the help of a model of the buffer's packet handling mechanism, we determine the mean time between the instant at which a packet is dropped and the time when three packets from the same flow are accepted. This helps estimate the time taken to trigger the fast retransmit and fast recovery procedures. The model only takes into effect the buffering delay, while the other delays are ignored.

Consider a system with N flows, in addition to a target flow, in a symmetric, time-slotted system. The slots are referred to by t, where t $\in \{0, 1, 2..\}$. Packets are assumed fixed in length and a packet transmission time is equal to one slot. The assumptions for the system, at any given time t, are listed below:

- Flows are symmetric, and each flow generates a packet in a slot according to a Bernoulli process with a probability p (asymmetric sources may be considered but at the expense of more computations).
- Let B be the total buffer size in the router. The service rate of the router is assumed to be 1 packet per slot.
- M_t is a random variable corresponding to the available buffer at the router at the beginning of slot t, and before accepting new packets.
- N_t is a random variable corresponding to the number of packets accepted by the router at the beginning of slot t.
- A_t is a random variable corresponding to the available buffer after packet acceptance at the beginning of slot t.
- S_t is a random variable corresponding to the number of packets served in slot t.

We assume that at $t = 0$ the buffer is full, and a packet is discarded from the target flow. We would then like to evalute the time to accept three other packets from the same flow, which will trigger three duplicate acknowledgements, and will cause the source to star the recovery procedure.

From the above definitions it can be determined that at time t

$$A_t = M_t - N_t.$$

Accordingly, the available buffer in the next time slot, $M_{t+1} = \min(A_t + S_t, B)$. The available buffer space at time t+1 can thus be calculated as

$$M_{t+1} = \min(M_t - N_t + S_t, B) \tag{2}$$

The service rate of the router in slot t is determined as follows

$$S_t = \begin{cases} 0 & \text{if } B - A_t < 1 \\ 1 & \text{if } B - A_t \geq 1 \end{cases} \tag{3}$$

The evolution of M_t and hence A_t, depend on the number of accepted packets within t, i.e., N_t. The probability of accepting N_t packets given M_t buffer spaces are available is given by

$$P(N_t = n_t | M_t = m_t) = \begin{cases} \binom{N}{n_t} p^{n_t} (1-p)^{(N-n_t)} & \text{if } n_t < m_t \\ \sum_{n=m_t}^{N} \binom{N}{n} p^n (1-p)^{(N-n)} & \text{if } n_t = m_t \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The probability of having M_{t+1} buffer spaces available at time t+1, given that M_t spaces were available at time t can then be evaluated using equation (5)

$$P(M_{t+1} = m_{t+1} | M_t = m_t) = \sum_{S_t} P(M_{t+1} = m_{t+1} | M_t = m_t, S_t = s_t)\ P(S_t = s_t | M_t = m_t) \tag{5}$$

The second term on the R.H.S. of (5) is given by

$$P(S_t = s_t | M_t = m_t) = \begin{cases} 1 & \text{if } s_t = 1 \text{ and } m_t < B \\ P(N_t > 0) & \text{if } s_t = 1 \text{ and } m_t = B \\ P(N_t = 0) & \text{if } s_t = 0 \text{ and } m_t = B \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The first term of equation(5) can be obtained using the relation shown in equation (2) as follows

$$P(M_{t+1} = m_{t+1} | M_t = m_t, S_t = s_t) = P(N_t = m_t + s_t - m_{t+1} | M_t = m_t, S_t = s_t) \tag{7}$$

Taking into account the different buffer occupancy cases, the right hand side of equation (7) can be evaluated as

$$P(N_t = n_t | M_t = m_t, S_t = s_t) = \begin{cases} P(N_t = n_t | M_t = m_t) & \text{if } m_t < B,\ s_t = 1, \text{ and } n_t \geq 0 \\ 1 & \text{if } m_t = B,\ s_t = 0 \text{ and } n_t = 0 \\ 0 & \text{if } m_t = B,\ s_t = 0 \text{ and } n_t > 0 \\ 0 & \text{if } m_t = B,\ s_t = 1 \text{ and } n_t = 0 \\ P(N_t = n_t | M_t = m_t) & \text{if } m_t = B,\ s_t = 1 \text{ and } n_t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Equation (7) can be evaluated by recursive computation of equation (8) with the aid of equations (3) to (6). We can also obtain the probability of m_t available buffer spaces at time t, given that there were no buffer spaces available at time zero. This can be used to evaluate $Q(t)$, the probability of accepting a packet from the target flow at time t, which can be expressed as

$$Q(t) = \sum_{m_t=1}^{B} P(M_t = m_t | M_0 = 0)\ [P(n_t < m_t) + \sum_{n=m_t}^{N-1} P(n_t = n) \times \frac{1}{n+1}] \tag{9}$$

The first term inside the summation in equation (9) is the probability of m_t available buffers at time t, given there were 0 buffers available at time 0. The target packet would be accepted if the number of

packets that arrive from all other flows, n_t, is less than or equal to m_t (with the probability expressed by the first term inside the square brackets) or the target packet is accepted among all the other packets, when the arrival rate is higher than the available buffer space (with the probability expressed by the second term inside the square bracket).

The probability of accepting the next packet from the target source, given that the previous packet was dropped at time 0, q, can be obtained from $Q(t)$ and the packet interarrival time, which is geometrically distributed with parameter p, namely

$$q = \sum_{t=1}^{\infty} Q(t)\ p \times (1-p)^{t-1} \tag{10}$$

Considering that a packet was dropped from the target flow at time t due to the buffer being full, let the time until next packet is accepted be T_1. The mean value of this time, $\overline{T_1}$, is equal to $\frac{1}{q}\ \overline{t}$. Also, the value of $\overline{t}$, the mean packet inter-arrival time, is expressed as$\frac{1}{p}$. Combining the two, we can obtain

$$\overline{T_1} = \frac{1}{pq} \tag{11}$$

Similarly, the time between accepting the first and second packet, $\overline{T_2}$ and the time between the second and third packet, $\overline{T_3}$ can be obtained. The derivation of $\overline{T_2}$ and $\overline{T_3}$ depends on the buffer availability when a packet is accepted. However, to be on the conservative side, we assume that the buffer is full once we accept a packet, and that the distribution of T_2 and T_3 is the same as that of T_1. Therefore, the total time for three packets to be accepted after a packet drop can be expressed as

$$\overline{T} = \frac{3}{pq} \tag{12}$$

Assuming a network with 10 sources, including the target source, i.e., $N = 9$, a buffer capacity of 50,

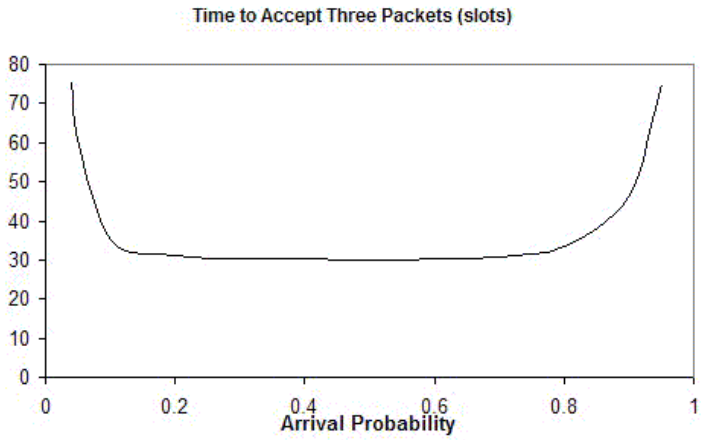

Figure 1: Graph showing the estimated time for three packets from the target source to be accepted, after a packet drop

the mean time to accept three packets in the router buffer after discarding a packet is plotted in Figure 1 for different values of p. This value is high initially as the packet arrival probability is low and so it takes longer for three packets to arrive. As the arrival probability increases, the time estimate decreases to a value of around 30 time slots and then again increases as the arrival probability exceeds 0.8. For example,

considering an arrival probability of 0.9, the time taken for recovery is over 40 time slots (in addition to the actual buffering delay) which can be significant, i.e., the flow would have already transmitted a number of packets.

The above provides the motivation for the schemes that are presented in this paper. The objective of these schemes is to provide mechanisms to trigger recovery from losses faster. Another objective is not to introduce any changes in the TCP protocol, which does not necessitate any changes to the end machines. Simple modifications will be required only at the routers. These modifications can be even made optional, which may or may not be enabled.

III.3 Scheme 1 - Triple Packet Buffering on Congestion

The idea behind this scheme is to make small changes to how the router (IP layer) handles packets during a congestion situation in which packets have to be dropped. The crux of this scheme is to attempt to trigger the fast retransmit and fast recovery as soon as a packet is to be dropped. This is done by avoiding the dropping of the three packets following a discarded packet.

Any packet to be dropped is uniquely classified according to the flow it belongs to on the basis of three attributes: the IP address of the source, the IP address of the destination and the protocol. Note that the flow is uniquely identified using a five tuple, which consists of the above three attributes in addition to the source and destination transport layer addresses. However, in order not to violate the layering concepts, the three attributes above are the only attributes used. The flows to which the schemes are attributed are those that use the TCP protocol. When a packet is to be dropped, the corresponding flow is registered in a table. This table entry maintains the number of times packets from such a flow are allowed to go through the buffer without dropping, after the flow has been registered in the table. When a new entry is made, this value is set to three as three subsequent packets from the same flow should not be dropped[1].

The next time a packet is to be dropped, the flow attributes of the packet are compared with the list of flows in the table. If it matches a flow that is already in the table, and if it has not yet used up all three chances of no dropping, then the packet is buffered instead of being discarded. For every such packet, the number of packets entry in the table is decremented by one. This causes three packets from the same flow to be sent across the network without being dropped, which in turn causes the destination to receive three packets from the source with sequence numbers that are higher than the current acknowledged sequence number, but with one segment missing before these. Since TCP provides cumulative acknowledgment, and only acknowledges data received in sequence, the receiver will send three duplicate acknowledgments for the three new packets received. When the sender receives the three duplicate acks, it enters the congestion recovery stage by invoking fast retransmit and fast recovery. The implicit assumption here is that the three new packets that arrive at the router, from the same flow as the dropped packet, have sequence numbers greater than the dropped packet.

A flow entry in the table is retained, even after all three packets have been sent across, and this persists for a fixed number of packets from the same flow, defined by *invocation interval*. The parameter *invocation interval* determines how often the proposed scheme is invoked on a packet, from a particular flow, that is selected to be dropped. After this, the flow entry is purged from the table. The flow is not removed immediately after the three no-drops because, the three no drops that went through would have already triggered a congestion recovery and if the flow were to be removed from the list immediately, the next immediate packet drop from the same flow will create another entry which would trigger the algorithm again, though the congestion notification has already been invoked. By allowing a gap of *invocation interval* packets we make sure that the current congestion scenario does not trigger multiple instances of the algorithm.

The pseudo code for scheme 1 is shown in figure 2. The parameter *packet_drop* is set when a packet is selected to be dropped by the packet drop scheme. The value of *invocation_interval* determines whether the current packet would be processed under the proposed scheme or not. *Clear_flow_entry()* is a routine that removes a flow, that has been through the scheme invocation, from the table. The packets are buffered in the routers queue by *queue_packet()* and *dont_drop* is a variable that keeps track of how many more packets

[1] This number of packets entry may be set to a larger value in order to allow for several scenarios, including the existence of multiple flows between the same pair of IP addresses, the use of delayed acknowledgements, misordered packet arrivals at the router, or the possibility of acknowledgement loss on the reverse path. In this exposition, however, we are not considering these cases since we are presenting the basic schemes.

```
if (packet_drop)
{
   if (flow_table_entry_exists)
   {
      if (dont_drop)
      {
         que_packet();
         dont_drop--;
      }
      else if (invocation_interval)
      {
         invocation_interval--;
      }
      else
      {
         clear_flow_entry();
      }
   }
   else
   {
      create_flow_entry();
      invocation_interval = INVOCATION_INTERVAL;
      dont_drop = 3;
   }
}
```

Figure 2: Pseudo code for scheme 1 - Triple Packet Buffering on Congestion

to buffer under the scheme.

One final comment on this scheme is that the packets from the flows which are entered in the table, may have to be given space priority over packets from other flows in order make sure they will arrive at the receiver. Since this may cause discarding packets from other flows, another option is to trigger discarding of packets before the buffer is actually full, similar to RED. We have chosen to implement the second option in our simulator.

III.4 Scheme 2 - Triple Truncated Packets

This scheme utilizes the same concepts as the previous scheme. Similar to the scheme in last section, the objective of this scheme is to trigger fast detection and recovery from congestion. However, in this scheme, instead of waiting for the next three packets to arrive, three shadow packets from the same flow are transmitted. However, such shadow packets should not be accepted by the receiver in order not to interfere with the actual data transfer.

Under this scheme, whenever a packet is to be dropped, instead of just dropping the packet, the packet is truncated and three copies of the truncated packet are created. These three packets are then enqueued to be routed to the destination. The truncation is done in such a way that, only the data part of the TCP packet is removed. The header information is preserved, which contains both the TCP and IP headers. The IP header checksum is recalculated, while the TCP checksum is not. When a truncated packet is received at the destination, it undergoes the TCP error checking procedure. As the data field has been removed, the checksum checking will indicate an error. This in turn triggers the transmission of a duplicate acknowledgment.

When all three truncated packets reach the receiver, they in turn trigger three duplicate acknowledgments which, when received by the sender, are interpreted as the presence of congestion in the network. The sender then quickly starts the fast retransmit and fast recovery procedures. As the packet transmitted is only 40

```
if (packet_drop)
{
   if (flow_table_entry_exists)
   {
        if (!invocation_interval)
        {
            clear_flow_table_entry();
        }
        else
        {
            invocation_interval--;
        }
   }
   else
   {
        clear_flow_entry();
        create_three_truncated_copies_of_pkt();
        queue_three_truncated_pkts();
        invocation_interval = INVOCATION_INTERVAL;
   }
}
```

Figure 3: Pseudo code for scheme 2 - Triple Truncated Packets

bytes, i.e., 20 bytes of IP header and 20 bytes of TCP header, the effect of these three duplicates on the router buffer holding capacity, as well as the transmission line utilization is minimal.

Similar to scheme 1, the same flow information is maintained in the table. Here too, once a packet from a flow triggers the algorithm, the next *invocation interval* number of packets that are to be dropped from the same flow do not trigger the algorithm.

The pseudo code for scheme 2 is shown in Figure 3. The parameters *packet_drop* and *invocation_interval* have the same functionality as in in Figure 2. Three truncated copies of the packet are created in *create_three_truncated_copies_of_pkt()*. The packets are then queued using the routine *queue_three_truncated_pkts()*. These routines are not shown here, but they are straightfoward. In order to provide an estimate of the efficacy of our two algorithms, we make use of the analytical model presented in Section III.2 to provide some performance results of these algorithms. Figure 4 shows the time it would take for each of the schemes to trigger three duplicate acknowledgements, and with the same parameters used in Figure 1. All the delays, other than the buffering delay, have been ignored in this graph. The time taken under standard implementation is the same as in Figure 1. Under scheme 1, the value is governed by the arrival rate. After a packet is dropped, three subsequent packets from the same flow are buffered. Therefore, the time taken is very high when the arrival rate is low and decreases as the arrival rate increases and would be very close to 3 for high loads. Scheme 2 truncates the packet, makes three copies and sends them. As the packet sizes are very small, about 40 bytes, this transmission can be done within one normal slot, so the time taken for triggering three duplicate acknowledgements in this case is 1, irrespective of the arrival probability. This graph underlines the benefits of our schemes. Further results based on simulation, and taking other delay and network parameters into account will be presented in the next section.

IV Performance evaluation

IV.1 Experimental Setup

The simulations were run on OPNET's Modeler simulator ([5]). The base scenario used in the experiments is shown in Figure 5. Three PPP sources are connected to a router, each through a DS3 link. The router itself is connected to a PPP server, which acts as the sink for all three sources. The link connecting the

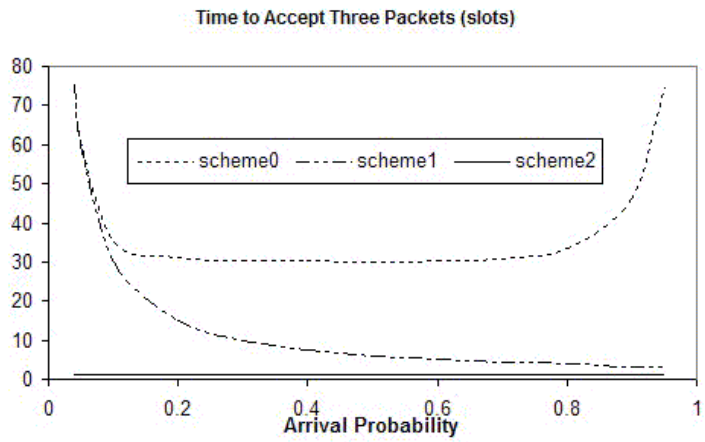

Figure 4: Graph showing the estimated time for triggering of three duplicate acknowledgements, under the three schemes

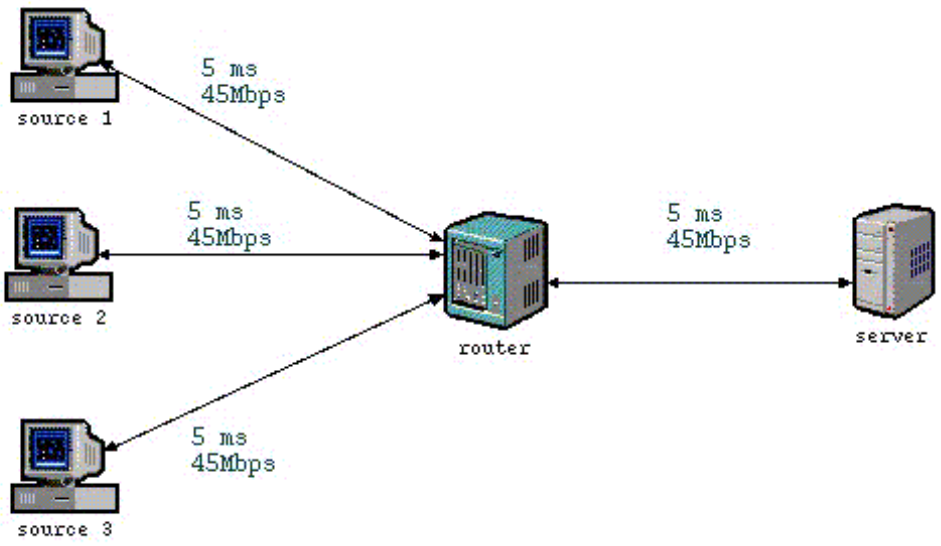

Figure 5: Base Network

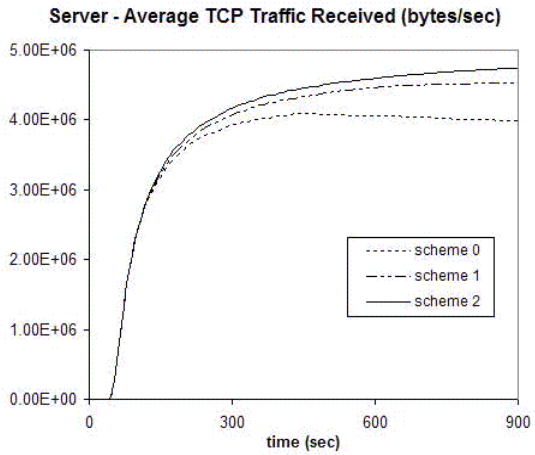

Figure 6: Comparison of Server TCP Throughput with *scheme0*, *scheme1* and *scheme2*

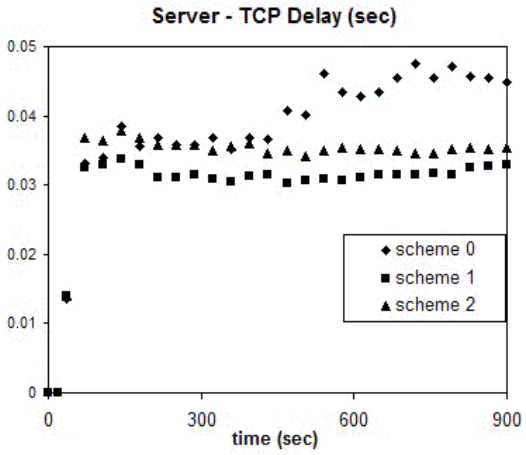

Figure 7: Comparison of Server TCP Delay between *scheme0*, *scheme1* and *scheme2*

	scheme0	*scheme1*	*scheme2*
Avg. Throughput(bytes/sec)	3991794.872	4522478.632	4750427.350
Avg. Delay(sec)	0.0394	0.0309	0.0346

Table 1: Result summary for base scenario at time = 900s

router and the server is also a DS3 link. All the links have a propagation delay of 5 ms. The sources are sending FTP traffic to the server, and it is assumed that each source has an endless supply of data, i.e., each source is restricted by the window size. The router buffer capacity is 50 packets. *Tail dropping* scheme is employed by the router for buffer management. The *invocation interval*, for the proposed schemes is set at 25. Results were collected for three different setups using this base scenario. The first one is the standard TCP Reno implementation, which will be referred to as *scheme0*. In the second case, the routers implement Triple Packet Buffering on Congestion, or *scheme1*. The third case, implements Triple Truncated Packets, *scheme2*, in the router. The simulation experiments were each run for 15 minutes.

IV.2 Homogeneous Networks

The case where all link propagation delays are the same, and the users' maximum window sizes are the same will be referred to as the homogeneous case. The results for this case are shown in Figure 6 and Figure 7. Figure 6 compares the Server TCP throughput of *scheme0* with *scheme1* and *scheme2*. Using the average value of throughput for comparison, the improvement over *scheme0* when using *scheme1* is 13.29% and in *scheme2* is 19.00%. The TCP throughput of the *router-assisted* schemes show marked improvement over the standard TCP Reno implementation. The improvements achieved by *scheme1* and *scheme2* are very similar, but they are more significant in *scheme 2*.

The TCP Delay measurements are compared in Figure 7. As with the throughput, the delay performance also improves in the proposed schemes compared to the standard TCP implementation. Compared to *scheme0*, the average value of the TCP delay decreases by 12.18% for *scheme1* and 21.57% for *scheme2*. The improvement in delay is attributed to the ability to recover from congestion expeditiously and that is due to the early congestion notification. The results for this section are summarized in Table 1, which shows the actual throughput and delay values.

The experiment was repeated with RED enabled routers with a max_{th} value of 50, min_{th} value of 17

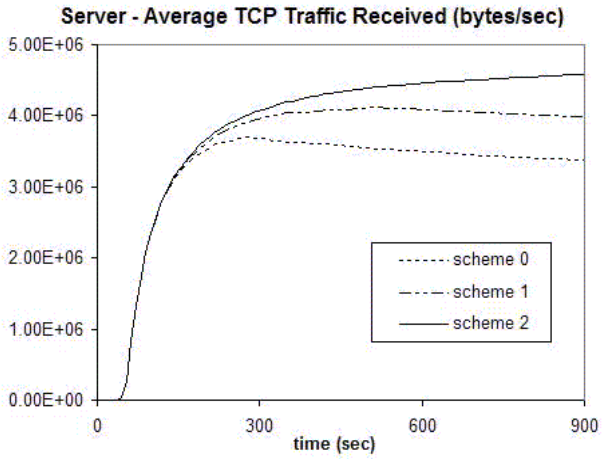

Figure 8: Comparison of Server TCP Throughput between *scheme0*, *scheme1* and *scheme2*, the router implementing RED.

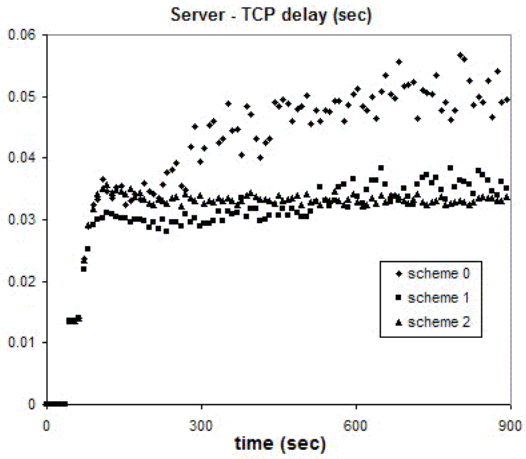

Figure 9: Comparison of Server TCP Delay between *scheme0*, *scheme1* and *scheme2*, with the router using RED packet drop scheme

	scheme0	*scheme1*	*scheme2*
Avg. Throughput(bytes/sec)	3382400.000	3992355.556	4580622.222
Avg. Delay(sec)	0.0437	0.0316	0.0326

Table 2: Result summary with RED-enabled routers at time = 900s

and p_a of 0.02. The Figure 8 compares the TCP throughput obtained under each scheme. As with earlier results, the throughput increases with the proposed scheme. The throughput improvement over *scheme0* is 18% in *scheme1*, and 35% in *scheme2*.

The comparison of the end to end delay at the TCP layer is shown in Figure 9. There is significant improvement in delay performance in both *scheme1* (27.67%) and *scheme2* (25.4%), as compared to *scheme0*. As discussed earlier, the proposed schemes expedite congestion notification, triggered by RED's proactive packet dropping. This explains the performance improvement, both in delay and throughput at the TCP layer. Table 2 summarizes the results for this section. Although not shown here, it should be pointed out that the throughput under some scenarios using RED enabled routers is lesser than with the same scenarios but with the DropTail packet drop scheme. This is similar to what has been observed in [7], as RED provides more overall fairness, although throughput can be reduced.

IV.3 Heterogeneous Networks

The above experiments were repeated but under a heterogeneous network scenario. In this case, sources 1 and 3 are connected to the router using DS1 links. The link between source 2 and the router operates at a DS3 speed. The comparison of the link throughput of sources 1 and 2 under the three schemes are shown in Figures 10 and 11, respectively. Source 2, which is the connection using a DS3 link, has a far higher transmission speed and therefore the number of packets transmitted is initially higher compared to the other sources, though the router bandwidth is far lower, at DS1. The window size of the TCP source also limits the maximum throughput. Thus, it suffers far more packet drops, and this causes its rate to be very low, performing worse than sources connected through DS1 links. The throughput comparison of the different schemes shows that the proposed schemes perform better than the standard implementation. However, with source 2, the throughput under *scheme1* drops very close to that of the standard implementation with time.

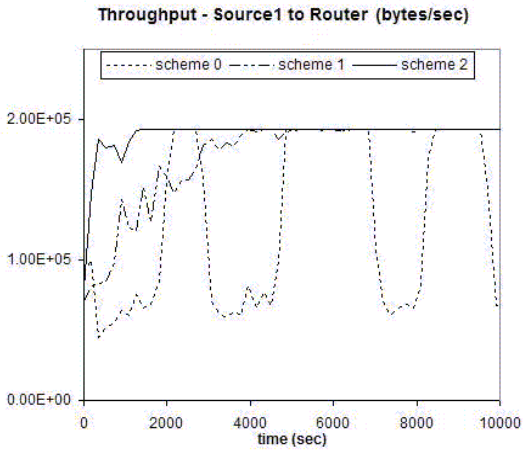

Figure 10: Comparison of throughput from source 1 (DS1 link) to router, with *scheme0*, *scheme1* and *scheme2*

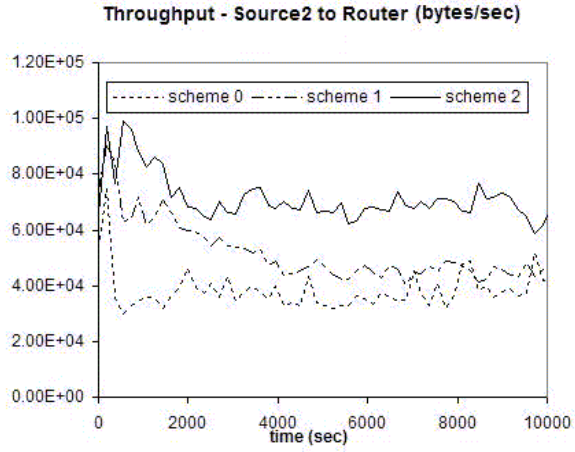

Figure 11: Comparison of throughput from source 2 (DS3 link)to router, with *scheme0*, *scheme1* and *scheme2*

A closer examination of the basic results in Figure 10 indicates that source 1 tries to ramp up the throughput at some stages and then there is sudden reduction in transmission. This indicates that there are multiple timeouts as there are not enough transmitted packets which can cause three duplicate acknowledgements due to persistent congestion. These timeouts trigger multiple slow starts and thus reduce the link utilization. The throughput values of *scheme1* and *scheme2* are maintained at full utilization, since the generation of duplicate acknowledgements is both enhanced, and expedited. This indicates that the proposed schemes are successful in keeping the TCP sources from going into slow start. Although the cumulative throughput at the TCP level was not improved substantially by this scheme, *router assisted* schemes are successful in ensuring that the TCP recovers through Fast Retransmit/ Fast Recovery, rather than going into Slow Start.

We also study the effect of links with differing link delays using the network shown in Figure 12. The delay of the link between source 1 and the router is 1ms and that between source 3 and the router is 9ms. The router to server link propagation delay is 5ms. The buffer capacity at the router is set to 15 packets and an *invocation interval* value of 7 is used.

The comparison of Source 1 throughput under the three schemes is shown in Figure 13. The throughput for both *scheme1* and *scheme2* are lower than *scheme0*. The throughput of the high delay link (i.e., from *source 2* and *router*, 9 ms), is shown in Figure 14. The throughput of the proposed schemes are higher than the standard implementation. This is not an anomaly, but rather a correction of an anomaly, as explained below.

It is known that the standard TCP implementation has a bias towards connections with low propagation speeds, as it takes less time to receive feedback about the network conditions. The *router assisted* schemes, by expediting the feedback mechanism, work towards reducing the variance in the throughput. Comparing Figures 13 and 14, it can be observed that this bias has been corrected through the reduced throughput for Source 1, and the increased throughput for Source 2. This results in a relative difference in throughput under the proposed schemes that is less than that under the standard implementation. The proposed schemes thereby are less biased towards difference in delays of the links.

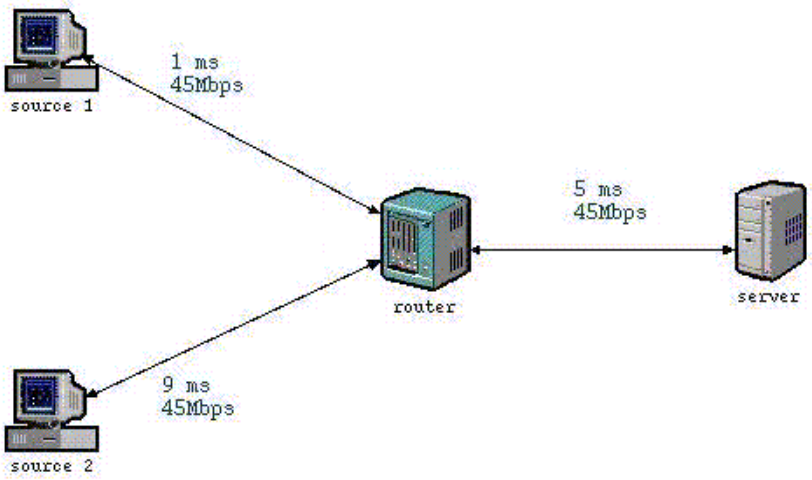

Figure 12: Heterogenous network with varying link delays.

V Conclusions

This paper has presented two schemes which expedite the detection of network congestion, and recover from packet losses in a timely fashion. The motivation behind these schemes is that in a typical network, network congestion, indicated by packet losses, is detected by retransmission timeout rather than three duplicate acknowledgements. The arrival of three duplicate acknowledgements can take a long time under heavy load due to multiple packet losses. It can also be long under light load, due to the long packet interarrival time. This was verified using a simplified analytical model of packet handling at the router.

The proposed, *router assisted*, schemes had the same objective, namely, that once a packet is to be discarded, the condition of three other packets arriving at the destination should be satisfied as soon as possible in order for the three duplicate acknowledgements to be generated, and for congestion to be detected. However, the two schemes differ in their operation. In the first scheme, the router allows at least three packets from the flow with the lost packets to propagate to the receiver. In the second scheme, however, artificial, short and erroneous packets are generated by the router, which does not require the arrival of three packets. Both schemes do not require any changes to the TCP protocol, and can be implemented easily at the routers.

It was shown through a simulation study that both of the proposed schemes allow TCP/IP networks to recover from congestion faster, and thus achieve a higher throughput, and lower mean delay. It has also been shown that the protocol behavior has been improved by avoiding multiple slow starts and timeouts. Also, the bias in TCP towards lower delay paths is also reduced under the proposed schemes. Hence, the two schemes are effective in enhancing the congestion control mechanism, without introducing any changes in the end-to-end, or network protocols.

References

[1] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. Network Working Group Request for Comment, RFC 2581, April 1999.

[2] S. Floyd. A report on some recent developments in TCP congestion control. *URL: "http://citeseer.nj.nec.com/floyd00report.html"*, June 2000.

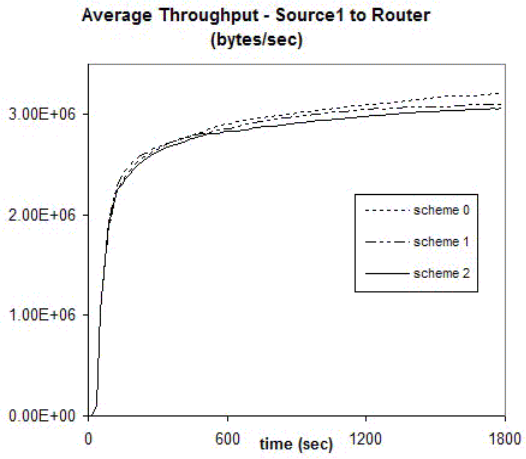

Figure 13: Comparison of link throughput, from source 1 to router, with *scheme0*, *scheme1* and *scheme2*

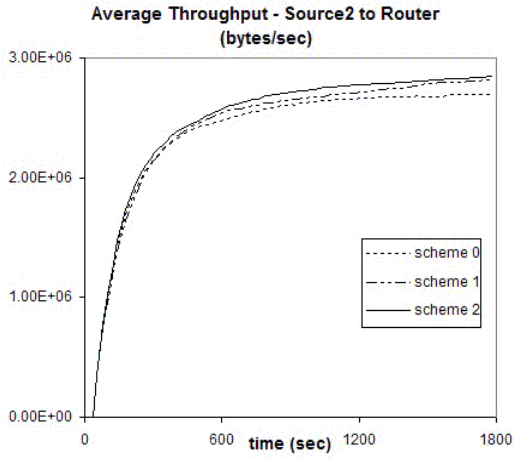

Figure 14: Comparison of link throughput, from source 2 to router, with *scheme0*, *scheme1* and *scheme2*

[3] S. Floyd and T. Henderson. The newreno modification to TCP's fast recovery algorithm. RFC2852, April 1999.

[4] J. Hoe. Improving the start-ip behavior of a congestion control scheme for TCP. In *ACM SIGCOMM'96*, pages 270–280, 1996.

[5] http://www.opnet.com.

[6] L. L. Peterson L. S. Brakmo, S. W. O'Malley. TCP vegas: new techniques for congestion detection and avoidance. In *ACM SIGCOMM'94*, pages 24–35, 1994.

[7] D. Lapsley and S. Low. Random early marking for internet congestion control. In *Global Telecommunications Conference, GLOBECOM'99*, pages 1747–1752, 1999.

[8] S. Floyd M. Mathis, J. Mahdavi and A. Romanow. TCP selective acknowledgment options. RFC2018, October 1996.

[9] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *ACM SIGCOMM'96*, pages 281–291, 1996.

[10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its emprical validation. In *ACM SIGCOMM'98*, pages 303–314, 1998.

[11] W. R. Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley, 1994.

[12] W. R. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. *RFC2001*, January 1997.

[13] H. Wang and K. G. Shin. Robust TCP congestion recovery. In *21st International Conference on Distributed Computing Systems*, pages 199–206, 2001.

[14] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2: The Implementation.* Addison-Wesley, 1995.