5

RFR FOR TCP APPLICATIONS

The Transmission Control Protocol (TCP) is the basic transport protocol for Internet applications such as email, web browsing, and file transfer. The majority of data traffic sent over the Internet is transported by TCP. IP networks are designed to be auto-controlled. The end station (host) implementing TCP adjusts its sending rate to the bandwidth of the path to the destination. The routers are in charge of topology changes in the network and of computing new paths based on the new topology. However, this mechanism does not ensure that the network runs in an efficient manner.

Although TCP is a reliable transport mechanism, packet loss, packet delay and packet disorder can seriously affect its performance, and hence application throughput. This is due to the TCP behavior of reducing its window size when congestion is detected or packet losses are detected, giving as a result a lower bandwidth for the application than the optimal one.

In this Chapter we evaluate the benefit for TCP applications of our proposal for a reliable and fast rerouting (RFR) mechanism described in Chapter 4. It is important to note that our work does not introduce any change in TCP. What the RFR does is to introduce a new function in the routers to take care of packet loss, packet disorder and packet delay for protected TCP traffic in an MPLS network (Chapter 4).

In the following section we present a brief explanation of TCP behavior and related algorithms.

5.1 OVERVIEW OF TCP BEHAVIOR

The Transmission Control Protocol (TCP) uses two main flow control mechanisms to transfer data from one end to the other. They are the receiver flow control and the sender flow control [tcp81] [Ste94].

The receiver flow control is based on the receiver using the ACK to advertise the allowable window size. In other words, when the receiver sends the acknowledgment packet to the sender it includes its available buffer size, indicating the amount of data that the sender can send at that time.

On the other hand, sender flow control uses the congestion window to manage its flow. The congestion window defines the maximum number of segments (in bytes) the sender can send without receiving an ACK. The sender's congestion window value is controlled and changed according to its implemented algorithm. Based on the above explanation, the amount of data the sender can send to the receiver without getting an ACK is the minimum of the receiver advertised window and the congestion window value (*cwdn*). Note that at any time during a TCP data transfer, either the receiver or the sender flow control is dominant.

There are four algorithms defined to control the congestion window during a TCP data transfer. The first two algorithms, *Slow Start* and *Congestion Avoidance* are

applied to all data transfers [Jac88]. The other two algorithms, *Fast Retransmit* and *Fast Recovery* are used when packet loss and packet reordering occur [Jac90].

5.1.1 Slow Start and Congestion Avoidance Algorithms

The slow start algorithm defines the way in which the *cwnd* is initially set and increases its value. When the TCP connection is established the congestion window is set to the default value, which is defined to be no more than two segments [APS99]. Then the sender increases its *cwnd* exponentially with each ACK received as an indication of the correct delivery of a sent segment. Thus, the value of the congestion window grows from one segment to two segments when the first segment is ACKed, and sends two segments. When it receives the ACK of these two segments it increases the *cwnd* from two to four. Thus, successively from four to eight, from eight to sixteen, and so on.

This exponential increment is limited either by the receiver advertised window (i.e., the amount of the data that the sender can send can not exceed the available buffer space at the receiver) or the limited capacity in any link along the path to handle the amount of data sent by sender. Observe that in this latter case, as the capacity of the path is not unlimited, if the sender continues increasing the sending rate it will cause packet losses. Packet loss will trigger the slow start algorithm. For this reason it is important to have a mechanism to control this exponential growth of the *cwnd* before the path capacity limit is reached. This is the point at which the congestion avoidance algorithm comes into effect to take the control of the *cwnd*.

The variable defined in [Jac88] to control the transition from slow start to congestion avoidance is called the *slow start threshold (ssthl)*. Congestion is assumed when the sender receives repeated ACKs of dropped or disordered packets, or when the time out is reached before receiving an ACK for a segment. When congestion is detected, the value of the *ssthl* is set to half of the current window size ($cwnd/2$) and the *cwnd* is set to one, forcing the sender window to the initial point of the slow start. Then

the sender starts the slow start algorithm, which functions as described before, as long as *cwdn*<= *ssthl*.

When *cwnd*> *ssthl* the congestion avoidance condition takes place: the incremental rate of *cwnd* is $1/cwnd$ for each arriving ACK, resulting a linear growth rate of one segment for each ACK. The congestion avoidance algorithm, as opposed to the exponential growth of the slow start, tries to provide the optimal size of *cwdn* for the delay-bandwidth product. However, despite this linear increment the sender can still reach the maximum capacity limit of any link on the path because the sender has no information about the link status. This condition again triggers the slow start, reducing the amount of data that can be sent through the path. The fast retransmission and fast recovery are proposed to alleviate this effect, using the link's maximum available capacity.

5.1.2 Fast Retransmit and Fast Recovery Algorithms

For the sender, the arrival of three consecutive repeated ACKs is an indication that the packet was actually dropped [APS99], triggering the retransmission of the segment. The retransmit occurs without waiting for a time out. This behavior is defined as fast retransmit [Jac90]. After fast retransmit takes place the sender enters into fast recovery. Note that in fast recovery the sender continues sending data at the optimal rate according the congestion avoidance algorithm until the arrival of the ACK for the retransmitted segment.

Unlike congestion avoidance, which reduces the *cwnd* to one, fast retransmit sets the *cwnd*=*ssthl*+ the number of the duplicated ACK, then continues receiving the ACK of those segments that were sent on the path before the fast retransmit came into action (i.e., retransmits the possibly dropped segment). For each ACK received in this period it increases the *cwnd* by one segment.

When the sender receives the ACK of the retransmitted segment it exits fast recovery mode, and the *cwnd* is set equal to the *ssthl* (i.e., half of the actual congestion window value). At this point congestion avoidance is initiated and the *cwnd* increases linearly according the congestion avoidance algorithm.

Fast recovery improves the throughput for a single packet loss, but multiple packet losses continue to affect the throughput because the window size (*cwnd*) decreases for each dropped packet. When used on high speed networks with a long delay and congestion, TCP becomes slow. When a TCP packet is lost, the sender must retransmit the packet, and the time it waits before retransmission increases according the delay, impairing TCP performance.

5.2 EVALUATION OF RFR FOR TCP CONNECTIONS

In order to evaluate the benefits of RFR for TCP applications during a link/node failure or congestion situation compared to Haskin's scheme we used the same simulation scenario 4.1.

For the throughput comparison for TCP traffic we setup an FTP session over a TCP connection with packet size = 1000 bytes. Figure 5.1 compares the behavior of RFR and Haskin's scheme.

In Figure 5.1 the difference in the sequence number of TCP segments received by the egress LSR is seen clearly for the same simulation time. Figure 5.2 shows a more detailed view of the sequence number during the restoration period.

Additionally, in Figure 5.2 the perturbation caused by the disorder of the packets can be seen (zoomed graph). Note that the time of link failure is 1.51 seconds. Figure 5.1 confirms that the proposed mechanism avoids packet loss and disordering. This benefit is due to the use of the buffer, which avoids the loss of packets and therefore the penalty due to retransmission.

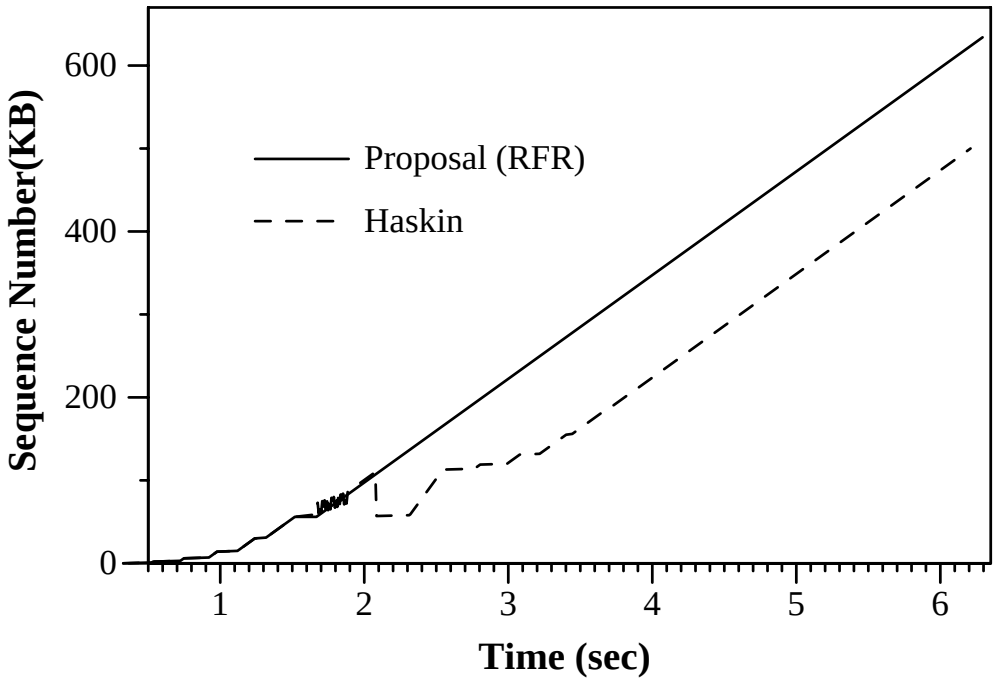

Figure 5.1 Behavior of TCP traffic for MSS of 1000 bytes

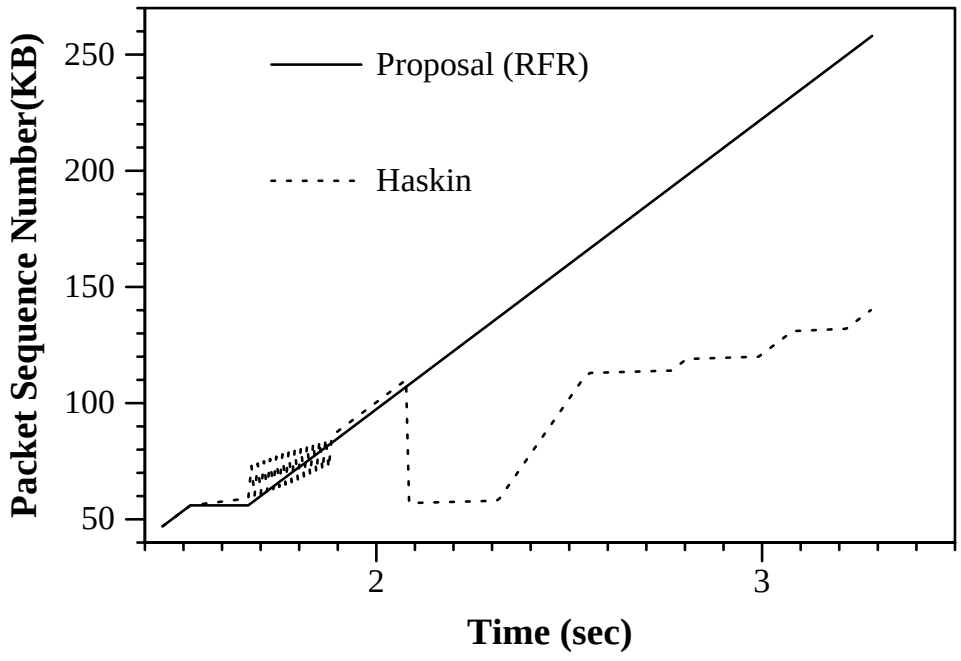

Figure 5.2 Behavior of TCP traffic for MSS of 1000 bytes

5.3 SUMMARY

As RFR avoids packet losses and packet disorder in the protected flows, TCP connections experience neither losses nor disordered packets, and can continue to run at the maximum throughput even during the restoration period of the protected LSP.