

TCP-R: A Protocol for Handling Out of Order Packets in Satellite Networks

Arjuna Sathiseelan and Tomasz Radzik
Department of Computer Science,
King's College London
Email: {arjuna,radzik}@dcs.kcl.ac.uk

ABSTRACT

Studies have shown that packet reordering is common, especially in satellite networks where there are link level re-transmissions and multipath routing. Moreover, satellite networks exhibit high corruption rates causing packet losses. Reordering and corruption of packets decrease the TCP performance of a network, mainly because it leads to overestimation of the congestion in the network. We consider satellite networks and analyze the performance of such networks when reordering and corruption of packets occurs. We propose a solution that could significantly improve the performance of the network when reordering and corruption of packets occur in a satellite network. We report results of our simulation experiments, which support this claim.

1. INTRODUCTION

TCP is the commonly used transport protocol in the Internet. The TCP protocol provides a reliable, connection oriented, in-order delivery of data between any two hosts in the Internet [16]. To ensure the data is delivered from the sender to receiver correctly and in-order, the TCP sender uses sequence numbers to each octet of data that is transmitted and the TCP receiver ACKs the receipt of these transmitted bytes that are received correctly and in-order. Since ACKs are cumulative, the receipt of out-of-order packets generate duplicate ACKs that are sent to the TCP sender. TCP assumes congestion in the network to be the cause of loss of packets. Thus when a TCP sender receives three successive duplicate ACKs, it assumes a packet has been lost and that this loss is an indication of network congestion and reduces its sending rate [10].

Satellites are currently being used as a medium of communication, to locations that lack adequate terrestrial infrastructure. Fleets of telecommunications satellites can provide network access to nearly every point on the globe. Broadband satellite constellation networks have been proposed at geostationary orbit (GEO), lower earth orbit (LEO), medium earth orbit (MEO) and highly elliptical orbit (HEO). These

satellite systems provide medium and high capacity wireless data services, and are interconnected with the existing terrestrial networks [7].

Networking using satellites began by using individual satellites in GEO, where the signals were amplified and then up-linked to the GEO satellite. The satellite then frequency-shifts the signal and broadcasts it down to a large ground area. These GEO satellites acted as simple transparent 'bent-pipe' repeaters [13]. A LEO network such as Iridium [12] has several satellites connected together to form a network. When there is congestion in a particular path, the satellite routes the packets through a different path. This introduces reordering of packets [19]. Satellite networks such as GEO and LEO have high RTTs, typically in the order of several hundred milliseconds. In order to keep the pipe full, link-layer retransmission protocols send subsequent packets while awaiting an ACK or NAK for a previously sent packet. Here, a link-layer retransmission is *reordered* by the number of packets that were sent between the original transmission of that packet and the return of the ACK or NAK [18].

In satellite networks, the packet loss is mainly due to corruption. These corrupted packets could be dropped either in the routers (in case of LEO network) or in the receiver when the header checksum fails. If the link layer at the receiver, detects any errors and if there is enough redundancy transmitted in the code, then the errors can be corrected using the Forward Error Correction algorithms without requesting for a retransmission. Therefore, the link layer could pass the error free packet to the top layers. If the link layer detects an error but cannot correct it (i.e. the cyclic redundancy check fails), the link layer drops the corrupted packet and the link layer at the receiver requests the link layer at the sender to retransmit the packet. These link level retransmissions only make a limited attempt to recover the lost packet. If the link layer cannot recover the lost packet, it will be left to the TCP layer in the worst case. Thus when packets are lost due to corruption, link layer protocols that do not attempt in-order delivery across the link cause packets to reach the TCP receiver in out-of-order. This leads to the generation of duplicate ACKs by the TCP receiver, which causes the sender to invoke fast retransmission and recovery [4].

There have been many proposals for extending TCP to improve the performance when the losses are mainly due to corruption. Some of the proposed mechanisms are the Explicit Loss Notification (ELN) mechanism [3], Explicit Transport

Algorithm	Description
DSACK-FA	DSACK-R + fixed FA ratio
DSACK-FAES	DSACK-FA + enhanced RTT sampling
DSACK-TA	DSACK-FA + Timeout Avoidance
DSACK-TAES	DSACK-TA + enhanced RTT sampling

Table 1: RR-TCP Algorithms

Error Notification (ETEN) [11], Indirect-TCP (I-TCP) [2], TCP PEACH [1] etc. These proposals improve the performance of TCP when packets get lost due to corruption in the network but do not consider the effects caused due to reordering of packets.

Several methods to detect the needless retransmission due to the reordering of packets have been proposed:

The DSACK option in TCP, allows the TCP receiver to report to the sender when duplicate segments arrive at the receiver’s end. Using this information, the sender can determine whether a retransmission was spurious [8]. If the retransmission was spurious, then the slow start threshold (*ssthresh*) is set to the previous congestion window (*cwnd*). Their proposal does not specify any mechanisms to proactively detect reordering of packets. We term this mechanism as DSACK-R (DSACK with Recovery) for testing purposes.

In [20], the authors propose mechanisms to detect and recover from false retransmits using the DSACK information. They propose several algorithms for proactively avoiding false retransmits by adaptively varying the duplicate threshold (*dupthresh*) value. The various algorithms used are listed in Table 1. In the DSACK-FA algorithm, the *dupthresh* value is chosen to avoid a percentage of false fast retransmits, by setting the *dupthresh* value equal to the percentile value in the reordering length cumulative distribution. The percentage of reordering the algorithm avoids is known as FA ratio algorithm. In the DSACK-FAES algorithm, the DSACK-FA algorithm is combined with a RTT sampling algorithm which samples the RTT of retransmitted packets caused by packet delays. The DSACK-TA algorithm uses cost functions that heuristically increase or decrease the FA ratio such that the throughput is maximized for a connection experiencing reordering. The FA ratio will increase when false retransmits occur and the FA ratio will decrease when there are significant timeouts. In the DSACK-TAES algorithm, the DSACK-TA algorithm is combined with a RTT sampling algorithm which samples the RTT of retransmitted packets caused by packet delays. According to [20], the DSACK-TA algorithm performed the best when compared with the other algorithms for various delay distributions. DSACK-TAES performed better than DSACK-TA for large packet delays that exceeded the one second minimum RTO. For other delay distributions, both DSACK-TA and DSACK-TAES perform similarly.

In [17], we proposed a novel method to enable the TCP senders to distinguish whether a packet has been dropped or reordered in the network by using the gateways to inform the ‘receiver’ about the dropped packets. This mechanism was called the Explicit Packet Drop Notification ver-

sion 2.0(EPDNv2.0). The receiver then uses this information to inform the sender about which packets have been reordered by setting a *drop-negative* bit. If the packets had been dropped in the network, the TCP sender retransmits the lost packets after waiting for three duplicate ACKs. If the packets are assumed to be reordered in the network, the TCP sender waits for ‘3+k’ duplicate ACKs ($k \geq 1$) before retransmitting the packets. We termed this new version of TCP as Reorder Notifying TCP (RN-TCP).

The proposals mentioned to alleviate the TCP performance in the presence of packet reordering do not consider error prone networks. It would be interesting to find out the performance of these protocols when reordering happens in a network that is error prone. If a packet had been actually dropped due to corruption, having an increased value of *dupthresh* may require a timeout to detect the packet loss. Thus increasing the *dupthresh* value to more than three when a packet has been assumed not to be dropped may have serious implications in the performance, if the packet had actually been dropped due to corruption. The EPDNv2.0 mechanism proposed by us in [17], informs the sender/receiver about dropped packets. This could be dropped due to congestion in the network or due to corruption in the network. RN-TCP retransmits the lost packet and reduce the transmission rate even if the packets had been dropped due to corruption. If a packet had actually been lost due to corruption, the performance of TCP can be improved, if the TCP sender does not reduce the *cwnd* upon a retransmission of the lost packet. Moreover, when networks exhibit high RTT, unnecessary reduction of the *cwnd* requires large number of RTTs to retrieve back to the previous *cwnd*. This reduces the throughput performance. Thus, it was imperative for us to propose a new TCP protocol, that can improve the throughput performance when packets experience reordering and corruption.

In this paper, we propose extending the TCP SACK protocol to enable TCP senders to recognize whether a received duplicate ACK means that a packet has been dropped or corrupted/ reordered. The extended protocol uses a mechanism called the Explicit Packet Drop Notification Version 3.0 (EPDNv3.0) mechanism to infer which packets have been dropped due to congestion. The TCP sender uses this information to take an appropriate action. We term this new TCP protocols as Robust TCP (TCP-R).

The remainder of this paper is organized as follows. Sections 2 and 3 presents the details of our proposed solution. In Sections 4,5,6,7,8 and 9, we describe and discuss the evaluations of our solution via simulations. We conclude the paper with a summary of our work and a short discussion of the further research in Section 11.

2. EPDNV3.0 FOR SATELLITE NETWORKS

We propose a mechanism similar to EPDNv2.0 (proposed by us in [17]), by maintaining information about packets dropped due to congestion in the gateways and not by header checksum error which occurs mainly due to corruption.¹ Each gateway maintains a hashtable, that records the max-

¹EPDNv2.0 maintains information about drops caused by both corrupted and congested packets

imum sequence number and minimum sequence number of the packets that get dropped in the gateway for each flow. When the next data packet of flow i passes through that gateway, the gateway inserts the maximum sequence number and the minimum sequence number of the dropped packets in the data packet and the entry is deleted from the data structure. We term this mechanism of explicitly informing the TCP receiver about the dropped information as Explicit Packet Drop Notification Version 3.0 (EPDNv3.0).²

3. TCP-R: ROBUST TCP

We propose extending the TCP SACK protocol to enable TCP senders to recognize whether a received duplicate ACK means that a packet has been dropped or corrupted/reordered in the network. The extended protocol uses the Explicit Packet Drop Notificationv3.0 (EPDNv3.0) mechanism to infer which packets have been dropped. The TCP-R receiver maintains two lists: the reorder/corruption list and the drop list. The TCP-R receiver uses the algorithm mentioned in Appendix A to determine which packets have been dropped due to congestion and which have been reordered or corrupted, and puts those sequence numbers into the corresponding lists.³ Informing the sender is done by setting the *drop-negative* bit in corresponding duplicate ACKs if the packet has been assumed to be reordered or corrupted.⁴ If the packets are assumed to be reordered or corrupted in the network, the TCP-R sender retransmits the packet after receiving three duplicate ACKs with the *drop-negative* bit set and enters our modified fast recovery mechanism where the procedure of reducing the *ssthresh* and the *cwnd* are bypassed i.e. we do not reduce the *ssthresh* and the *cwnd*. In order to prevent TCP-R from falsely misjudging drops due to congestion and not reducing the sending rate, we ensure the modified fast recovery mechanism is executed only in the absence of ECN messages. If the packets had been dropped in the network, the TCP-R sender retransmits the lost packet after waiting for three duplicate ACKs (fast retransmit) and reduces the *cwnd* by half (fast recovery).⁵

In an environment where out-of-order packets can happen due to corruption and reordering, the throughput performance can be improved if we do not reduce the *cwnd* upon the occurrence of a reordering or a corruption event. Even though TCP-R unnecessarily retransmits a reordered packet, it reduces the instances of timeouts by not delaying the retransmission of corrupted packets. Moreover it does not reduce the *cwnd* unnecessarily, thus improving the throughput of the sender.

3.1 Sender side: Implementation Details

When an ACK is received, the TCP sender does the following,

- If none of the three duplicate ACKs received have their

²The implementation details are similar to EPDNv2.0. The implementation issues and costs have been thoroughly analyzed in [17].

³TCP-R uses the reorder list specified in [17] to store out-of-order packet sequence numbers caused by losses due to corruption and reordering. We denote the list as reorder/corruption list

⁴The *drop-negative* bit is set to 1

⁵The *drop-negative* bit is set to 0

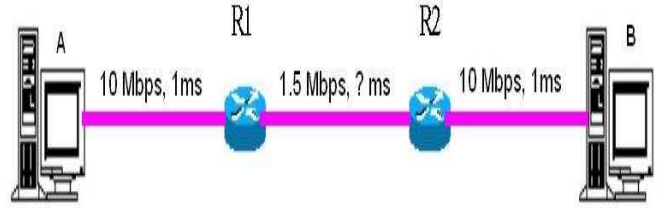


Figure 1: TCP-R: Simulated Network

drop-negative bit set, then the sender assumes that the packet has been dropped. So the sender retransmits the lost packet after receiving three duplicate ACKs and enters fast recovery.

- If all the three duplicate ACKs received have their *drop-negative* bit set and the ECE bit is set to zero (i.e No ECN information has been received off lately), then the sender assumes that the packet has been reordered or corrupted in the network and retransmits the packet immediately. The procedure of reducing the *ssthresh* and the *cwnd* in the fast recovery procedure are bypassed. In order to avoid multiple retransmits, we ensure that a packet assumed to be reordered or corrupted would be retransmitted only once in a RTT.

4. SIMULATION ENVIRONMENT

Figure 1 presents the topology used for our simulations. The simulated network has a source and destination node connected to two intermediate routers. The nodes are connected to the routers via 10 Mbps Ethernet having a delay of 1 ms. The routers are connected to each other via long delay link with a fixed link capacity and variable delay. Our simulations use 1500 byte packets. We used the RED queueing strategy with a queue size set to the bandwidth delay product. All routers were ECN enabled. Reordering and packet drops are introduced at the bottleneck link (R1,R2). The experiments were conducted using a single long lived FTP flow traversing the network topology, except otherwise noted. The maximum window size of the TCP flow was also set to the bandwidth delay product. The TCP flow lasts 1000 seconds.

Our simulations consider the case of both LEO and GEO satellite links. The lack of flexibility of the current ns-2 simulator to delay a fraction of packets and to test for various average packet delays for satellite links caused us to model the satellite links by representing a wired link with the same capacity and delay as a GEO or a LEO satellite link, similar to [9]. The GEO satellite link has roughly a delay of 300 ms (one way). The LEO satellite link has a one way delay that varies [40,400] ms depending on whether the LEO network has one satellite hop or multiple hops and how far each of these satellite hops are placed [9]. The link capacity was set to 1.5 Mbps (T1 carrier). In GEO networks, we consider the reordering to be caused only due to link level retransmissions and in LEO networks we consider the case of link level retransmissions and multipath routing. Reordering is caused when a delayed packet with a higher sequence num-

ber is scheduled to traverse the link later than an un-delayed packet with a lower sequence number. To simulate packet reordering, we delay a percentage of packets traversing the link by delay distributions. Our experiments can be classified into the following scenarios.

- *Scenario 1.* The link layer at the receiver could detect and correct *some* of the corrupted packets using link level retransmissions. Those packets that cannot be recovered at the link layer, will not arrive at the TCP, causing out-of-order packets. In addition to this, some corrupted packets may have been dropped by the satellite nodes (incase of LEO networks). Due to retransmission at the link layer, some of the packets could have been reordered. Thus the TCP receiver will receive out-of-order packets caused by both reordering and corruption. In order to simulate packet loss due to corruption, the experiments were performed for a BER of 10^{-6} .
- *Scenario 2.* The link layer at the receiver could detect and correct *all* corrupted packets using link level retransmissions, but these link level retransmissions could cause reordering. In this scenario, we assume all packets are correctly retrieved in the link layer after retransmissions. In addition to this, we also assume that the intermediate satellite nodes do not drop any corrupted packets. In order to simulate no loss of packets, we set the BER to zero. Thus the TCP receiver will receive out-of-order packets caused by reordering only.
- *Scenario 3.* Reordering of packets caused due to multipath routing (in LEO networks) and some packets getting dropped due to corruption. Thus the TCP receiver will receive out-of-order packets caused by reordering and corruption. In order to simulate packet loss due to corruption, the experiments were performed for a BER of 10^{-6} .
- *Scenario 4.* Reordering of packets caused due to multipath routing (in LEO networks) and no packets getting dropped due to corruption. In order to simulate no loss of packets, we set the BER to zero. Thus the TCP receiver will receive out-of-order packets caused by reordering.

We compare TCP-R with RN-TCP(EPDNv2.0) i.e when RN-TCP operates with EPDNv2.0, RN-TCP(EPDNv3.0) i.e. when RN-TCP operates with EPDNv3.0 specified for satellite links in Section 2, DSACK-TA, DSACK-R and SACK.

5. RESULTS - REORDERING DUE TO LINK LEVEL RETRANSMISSIONS

In this section, we consider the case when packets get reordered due to link level retransmissions in GEO and LEO networks. The propagation delay was set to 300 ms. To introduce severe packet delays in the order of multiple of RTTs, we used a mean packet delay of yP s (P is the propagation delay) and standard deviation of $\frac{y}{3}P$ s, such that the delay introduced varied from 0 to $2yP$ s. The packet delay rate was fixed at 4%. We varied the value of y from 1.0 to 6.0.

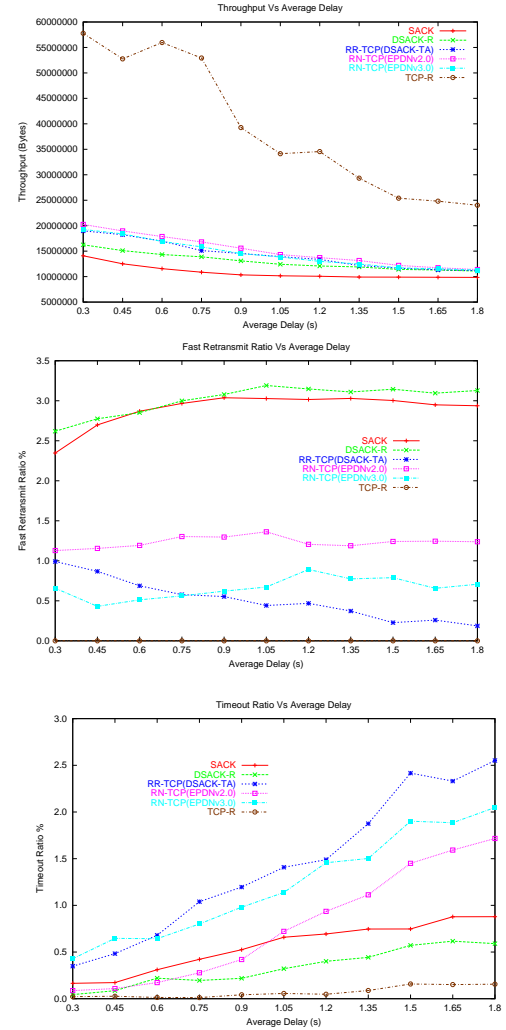


Figure 2: GEO link. Propagation delay of 300 ms. BER of 10^{-6} .

5.1 Throughput: Large Reordering Delays (Scenario 1)

In this section, we compare the throughput performance of TCP-R, RN-TCP(EPDNv2.0), RN-TCP(EPDNv3.0), DSACK-TA, DSACK-R and SACK when the packets experience reordering and losses due to corruption. In order to simulate packet losses due to corruption, we set the BER to 10^{-6} . Figure 2 presents the results of the simulations. As we increase y , the RTT of the packet that gets delayed exceeds the one second minimum RTO causing large number of timeouts. TCP-R outperforms RN-TCP(EPDNv2.0), RN-TCP(EPDNv3.0), DSACK-TA, DSACK-R and SACK for all tested mean packet delays from 0.3 s to 1.8 s. For example, when the average packet delay is 0.9 s, TCP-R gives a three fold throughput improvement over the other protocols. When the average packet delay is 1.8 s ($3 \times RTT$), TCP-R gives more than a two fold throughput improvement over the other protocols.

When packets get dropped in the gateways due to header

checksum error caused by corruption, EPDNv2.0 informs the sender about the dropped information. RN-TCP(EPDNv2.0) on receipt of this information, retransmits the corrupted packet immediately and reduces the *cwnd* after receiving three duplicate ACKs. Thus RN-TCP(EPDNv2.0) encounters more fast retransmissions than RN-TCP(EPDNv3.0) but lesser incidence of timeouts compared to RN-TCP(EPDNv3.0) and DSACK-TA. On the other hand, EPDNv3.0 informs the sender only about packets dropped due to congestion. Thus RN-TCP(EPDNv3.0) would assume the packet to be re-ordered and delays the fast retransmission procedure. This could cause the timer to expire leading to timeouts. Unlike RN-TCP and DSACK-TA, TCP-R on detecting the packet has not been dropped due to congestion in the network, retransmits the packet after receiving three duplicate ACKs without reducing the *cwnd*. This reduces the incidence of timeouts and unnecessary reduction of the *cwnd* due to fast retransmissions, leading to an improved throughput performance.

5.2 Throughput: Large Reordering Delays (Scenario 2)

Figure 3 presents the results of the simulations when packets just experience reordering and no loss of packets due to corruption. RN-TCP(EPDNv2.0) and RN-TCP(EPDNv3.0) perform similarly as both EPDN versions store only packets that get dropped due to congestion as there are no drops due to corruption. Initially, TCP-R performs slightly less compared to RN-TCP and DSACK-TA for average packet delays of 0.3 s and 0.5 s. From further investigation, as TCP-R sends more packets compared to RN-TCP and DSACK-TA (TCP-R does not prevent retransmission of corrupted or reordered packets), the ECN enabled routers mark the packets of the TCP-R flow by setting the CE bit. This causes TCP-R to reduced the *cwnd* more often. When the average packet delay is more than 0.6 s, TCP-R performs better than the other protocols. For example, when the average packet delay is set to 0.9 s, TCP-R gives a 4% throughput improvement over RN-TCP, 9% improvement over DSACK-TA, four times more than DSACK-R and six times more than SACK. Similarly when the average packet delay is set to 1.8 s, TCP-R gives a two fold throughput performance over RN-TCP, 80% improvement over DSACK-TA, three times more than DSACK-R and SACK. Moreover DSACK-TA undergoes large number of *cwnd* reductions due to fast retransmissions and has a higher timeout ratio compared to RN-TCP and TCP-R. TCP-R has almost zero fast retransmit and timeout ratios.

6. RESULTS - MULTIPATH ROUTING

In this section, we consider the case when packets get re-ordered due to multipath routing in LEO networks. The propagation delay was set to 400 ms. Multipath routing produces modal delays i.e. when successive packets are sent on paths with different RTTs, these packets would be re-ordered proportionally to the RTT difference of the path. 50% of the data packets were delayed. We performed experiments by gradually increasing the average packet delay from 0.0 s to 0.8 s ($2 \times RTT$). When the average packet delay is 0.0 s, the packets are routed through the same path without any reordering events.

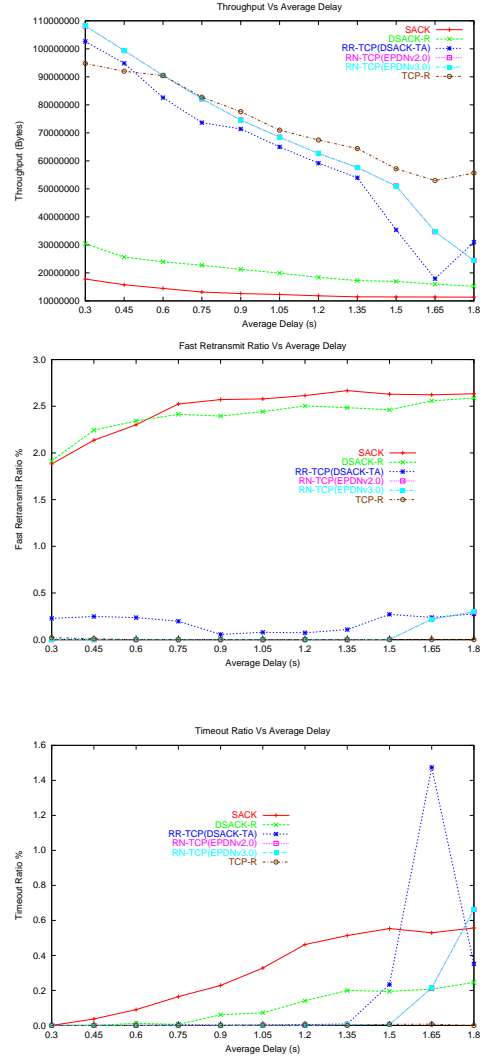


Figure 3: GEO link. Propagation delay of 300 ms.

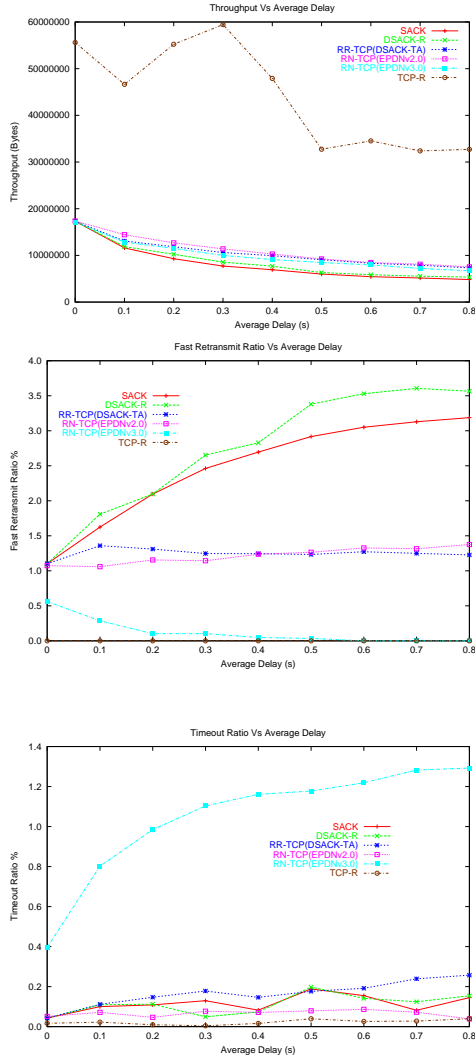


Figure 4: LEO link - Multipath performance, propagation delay of 400 ms. BER of 10^{-6} .

6.1 Throughput: Multipath Routing (Scenario 3)

In this section, we compare the throughput performance of the simulated network using SACK, DSACK-R, DSACK-TA, RN-TCP and TCP-R when packets experience losses due to corruption and reordering. The experiments were performed for a BER of 10^{-6} .

Figure 4 presents the results of the simulations when the propagation delay was set to 400 ms. For all tested average packet delays, TCP-R outperforms the other protocols. For example, when the average packet delay is 0.4 s, TCP-R performs four times more than RN-TCP(EPDNv2.0), five times more than DSACK-TA and RN-TCP(EPDNv3.0), six times more than DSACK-R and almost seven times more than SACK. Similarly, when the average packet delay is 0.8 s, TCP-R performs four times more than RN-TCP(EPDNv2.0), RN-TCP(EPDNv3.0), DSACK-TA, six times more than DSACK-R and almost seven times more than SACK. RN-TCP(EPDNv2.0) compare the throughput performance of the protocols when

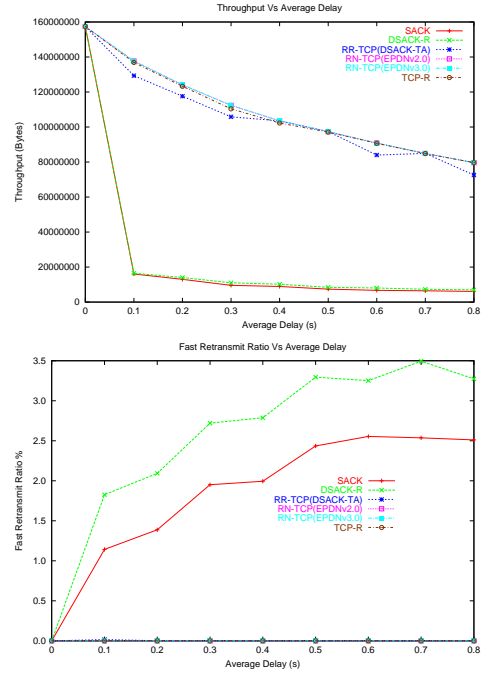


Figure 5: LEO link - Multipath performance, propagation delay of 400 ms.

encounters fewer timeouts compared to RN-TCP(EPDNv3.0) and DSACK-TA. TCP-R maintains a low incidence of timeouts and maintains a zero fast retransmit ratio compared to the other protocols, thus giving a better throughput performance.

6.2 Throughput: Multipath Routing (Scenario 4)

In this section, we compare the throughput performance of the simulated network using SACK, DSACK-R, DSACK-TA, RN-TCP and TCP-R when packets experience reordering due to multipath routing and no drops due to corruption.

Figure 5 presents the results of the simulations when packets experience reordering. TCP-R performs almost similar to RN-TCP for all tested average packet delays. TCP-R gives a better throughput performance when compared to DSACK-TA for majority of the average packet delays. For example when the average packet delay is 0.2 s, TCP-R gives a 5% throughput improvement over DSACK-TA and when the average packet delay is 0.8 s, TCP-R gives a 10% improvement in throughput performance when compared to DSACK-TA. TCP-R outperforms DSACK-R and SACK for all tested average packet delays. None of the protocols exhibit any timeouts. TCP-R and RN-TCP maintain a zero fast retransmit ratio. DSACK-TA's fast retransmit ratio hovers around zero. DSACK-R and SACK have a higher fast retransmit ratio.

6.3 Varying delay with constant bandwidth

In order to analyze the performance of the protocols over the range of propagation delays exhibited by LEO networks, we compare the throughput performance of the protocols when

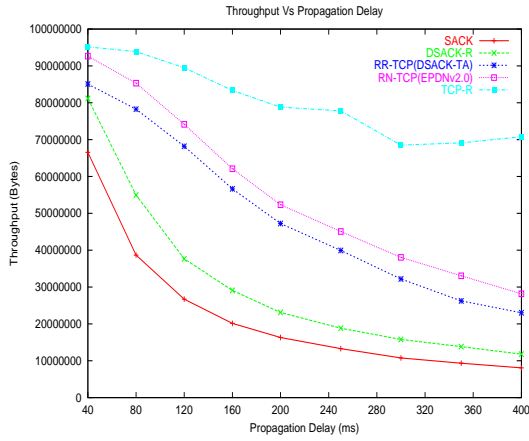


Figure 6: Throughput versus propagation delay.

the propagation delay varied from $[40,400]$ ms. The bandwidth was fixed at 1.5 Mbps. 7% of packets were delayed using uniform distribution $[0, 4P]$ where P is the propagation delay. The BER was set to 10^{-7} .

As shown in the Figure 6, it is evident that TCP-R outperforms SACK, DSACK-R RN-TCP and DSACK-TA irrespective of the propagation delay. For large propagation delays, TCP-R gives an improved throughput performance. For example when the propagation delay was set to 80 ms, TCP-R gives a 10% throughput improvement over RN-TCP, 20% improvement over DSACK-TA, 71% more than DSACK-R and a two fold throughput performance over SACK. When the propagation delay was set to 200 ms, TCP-R gives a 50% improvement over RN-TCP, 66% improvement over DSACK-TA, three fold improvement over DSACK-R and four times more than SACK.

7. VARYING BIT ERROR RATES

In this section, we compare the throughput performance of the protocols for various BERs. The bandwidth was fixed at 1.5 Mbps. The propagation delay was fixed at 300 ms. 5% of packets were delayed using normal distribution with a mean packet delay of 0.6 s and a standard deviation of 0.2 s such that the packets were delayed between 0 and 1.2 s. We tested the protocols for various BERs from a relatively low BER of 10^{-13} to a relatively high BER of 10^{-5} .

From the Figure 7, it is evident that TCP-R gives a better throughput performance compared to the other protocols for different BERs from 10^{-13} to 10^{-6} . When the BER is 10^{-5} , all protocols give a similar throughput performance. For BERs from 10^{-13} to 10^{-8} , TCP-R, DSACK-TA, RN-TCP(EPDNv2.0) and DSACK-R have almost a zero timeout ratio, whereas SACK encounters more timeout event and thus has a large timeout ratio. For BERs of 10^{-7} and 10^{-6} , TCP-R still maintains a zero timeout ratio whereas the other protocols start encountering more timeout events and have a larger timeout ratio compared to TCP-R. When the BER is large as 10^{-5} , all protocols undergo lots of timeout events and thus have a large timeout ratio. Even though RN-TCP(EPDNv2.0) has a large fast retransmit ratio com-

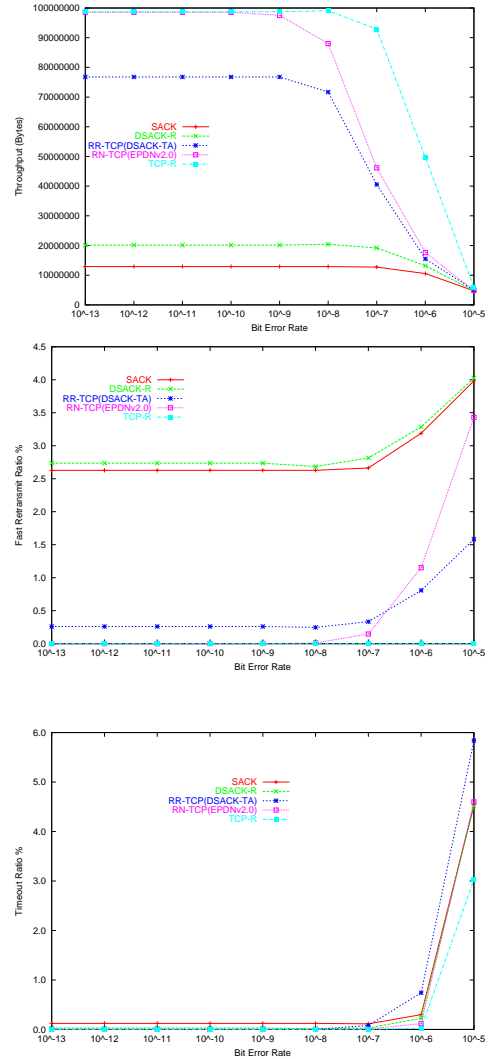


Figure 7: Performance for various BERs.

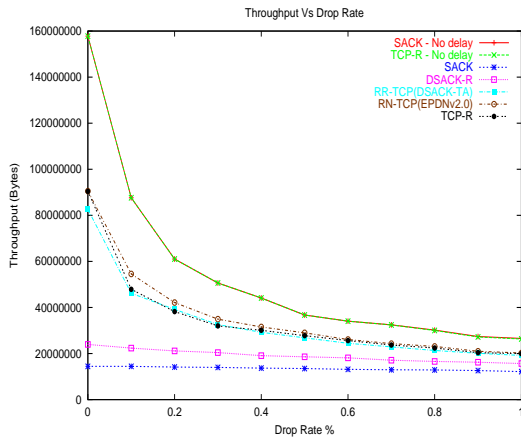


Figure 8: Throughput versus packet drop rate.

pared to DSACK-TA, DSACK-TA experiences more timeout events compared to the RN-TCP(EPDNv2.0). Thus RN-TCP(EPDNv2.0) performs better compared to DSACK-TA. TCP-R has a zero fast retransmit ratio and thus gives an overall improvement in throughput compared to the other protocols.

8. THROUGHPUT: PACKET DROPS DUE TO CONGESTION

In this section, we compare the throughput performance of the protocols when the link experiences both packet delays and packet drops due to congestion in the network. We also compared the performance of SACK and TCP-R with packet drops only. The propagation delay was set to 300 ms. 4% of the packets were delayed with a mean packet delay of 0.6 s and a standard distribution of 0.2 s such that the packets were delayed between 0 s to 1.2 s. The packet drop rate varied from 0% to 1%. We assumed that there were no loss of packets due to corruption and thus we set the BER to zero.

Figure 8, reveals that the throughput of all the protocols reduce considerably when packets get dropped. When packet drops occur, the throughput of any TCP variant would reduce drastically even when there is no reordering in the network. This is evident from the graph, where the performance of SACK with no delay reduce drastically with increasing packet drops. Moreover, our TCP-R with no delay performs similar to SACK with no delay. Thus, it is clear that given there are no reordering events, TCP-R performs similar to SACK. When packets get dropped, TCP-R performs almost similar to DSACK-TA. RN-TCP is able to achieve a better throughput when compared to TCP-R and DSACK-TA. But when the packet drop rate is increased from 0.4%, the performance of TCP-R, RN-TCP and DSACK-TA are similar. It is to be noted that all protocols experience similar throughput performance when large number of packets get dropped in the network due to congestion in the network.

9. MULTIPLE FLOWS

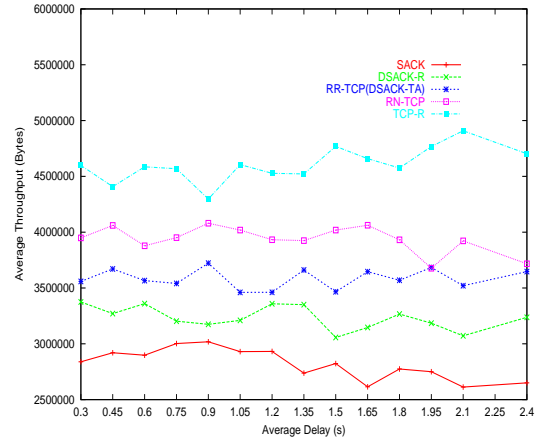


Figure 9: Average throughput versus average packet delay.

9.1 Performance: ECN enabled multiple FTP flows

In this section, we conducted a simulation on the same topology as in Figure 1, but used RED queues instead of *drop-tail* queues. The gateways were ECN enabled with a queue size set to 300 packets. The link delay was set to 300 ms. To introduce severe packet delays, we used a mean of yP s (P is the propagation delay) and standard deviation of $\frac{y}{3}P$ s, such that the delay introduced varied from 0 to $2yP$ s. The packet delay rate was fixed at 4%. We varied the value of y from 1.0 to 8.0. The BER was set to 10^{-6} . The total number of flows traversing the network were increased to fifty flows, in which the sender was configured to send ten SACK flows, ten DSACK-R flows, ten DSACK-TA flows, ten RN-TCP flows and ten TCP-R flows. All the flows were enabled with ECN. However, when the number of flows are increased, congestion will be caused in the bottleneck queue causing large number of packet drops. The graphs in Figure 9 present the results of the average throughput of SACK, DSACK-R, DSACK-TA, RN-TCP and TCP-R flows. It can be seen from the graph, that for all average packet delays from 0.3 s to 2.4 s, TCP-R outperforms the other protocols. For example, when the average packet delay is 0.3 s, TCP-R gives a 16% improvement over RN-TCP, 29% improvement over DSACK-TA, 36% improvement over DSACK-R and a 62% improvement over SACK. Similarly when the average packet delay is 2.4 s, TCP-R gives a 26% improvement over RN-TCP, 29% improvement over DSACK-TA, 45% improvement over DSACK-R and a 77% improvement over SACK.

10. CONSERVATIVE RTO ESTIMATION

DSACK-TA and RN-TCP avoids sampling RTTs for all packets that have been retransmitted by timeouts or fast retransmit, in accordance with Karn's algorithm [10], since the sender cannot infer whether an ACK matches an original transmission or the retransmission of a data packet. In the case of paths that predominantly delays packets, it is the delayed packets that are most likely to provoke retransmissions. Thus they are not included while estimating the RTO value. This produces RTO estimates that are too short lead-

ing to false timeouts. This is evident from the poor throughput performance achieved by RN-TCP and DSACK-TA for large average packet delays in the previous sections. Thus it is imperative to implement a conservative RTO sampling algorithm that considers both transmission and retransmission of data packets. DSACK-TAES is a combination of DSACK-TA and a conservative RTO sampling algorithm and [20] proves that for large packet delays, DSACK-TAES performs much better compared to DSACK-TA. In order to verify the performance of DSACK-TAES with our proposed protocols, we also ensure that our RN-TCP algorithm is also enhanced with this new RTO sampling algorithm.

When packets are falsely retransmitted, either by fast retransmit or by a timeout, two ACKs return, the second of which is a DSACK. When the sender receives both these ACKs, it can calculate the RTTs experienced by the packets by pairing the first ACK with the first transmission, and the second ACK with the second transmission. It can then calculate the time elapsed for each and take the mean of these two values as a single RTT sample for the RTO estimator. The scoreboard data structure used by SACK can hold this time information associated with the packet's transmission and retransmission. In case no DSACK arrives at the sender, then either the original or retransmitted packet was lost due to packet drops, and we do not sample that packet's RTT. This complies with the Karns Algorithm for retransmissions caused by packet drops, but include additional RTT samples for retransmissions caused by packet delays. We term this RTT sampling extension to RN-TCP as RN-TCP-ES (Reorder Notifying TCP with Enhanced RTT Sampling).

10.1 Throughput: Large Reordering Delays (Scenario 1)

In this section, we analyze the throughput performance of TCP-R over RN-TCP, RN-TCP-ES, DSACK-TA and DSACK-TAES when packets experience both reordering due to link level retransmissions and packet losses due to corruption. In order to drop packets based on corruption, the BER was set to 10^{-6} . Figure 10 presents the results of the simulations when the propagation delay was set to 300 ms. To introduce severe packet delays, we used a mean packet delay of yP s (P is the propagation delay) and standard deviation of $\frac{y}{3}P$ s, such that the delay introduced varied from 0 to $2yP$ s. The packet delay rate was fixed at 7%. We varied the value of y from 1.0 to 7.0. As we increase y , the RTT of the packet that gets delayed exceeds the one second minimum RTO causing large number of timeouts. For all tested average packet delays from 0.3 s to 0.9 s, the performance of RN-TCP-ES and DSACK-TAES is almost similar to RN-TCP and DSACK-TA respectively. When the average packet delay exceeds the one second minimum RTO, the performance of RN-TCP and DSACK-TA begin to decrease whereas RN-TCP-ES and DSACK-TAES maintain a steady throughput. For all tested average packet delays, TCP-R outperforms the other protocols. For example, when the average packet delay is 0.9 s, TCP-R gives a three fold throughput improvement over the other protocols. Similarly, when the average packet delay is 1.8 s, TCP-R gives a 62% throughput improvement over RN-TCP-ES, 81% throughput improvement over RN-TCP, 77% throughput improvement over DSACK-TAES and almost a 86% improvement over DSACK-TA. TCP-R maintains a low timeout ratio and a zero fast retransmit ratio compared to

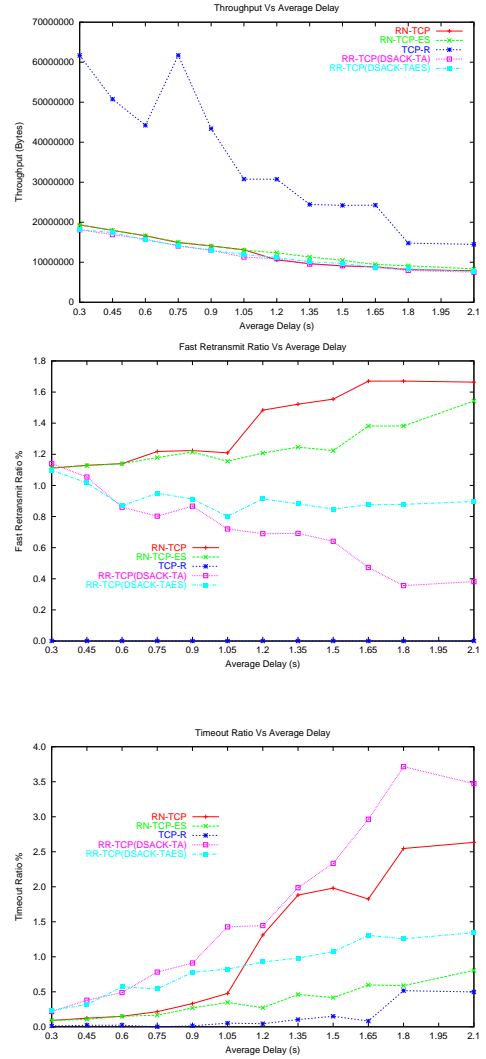


Figure 10: Propagation delay of 300 ms. BER of 10^{-6} .

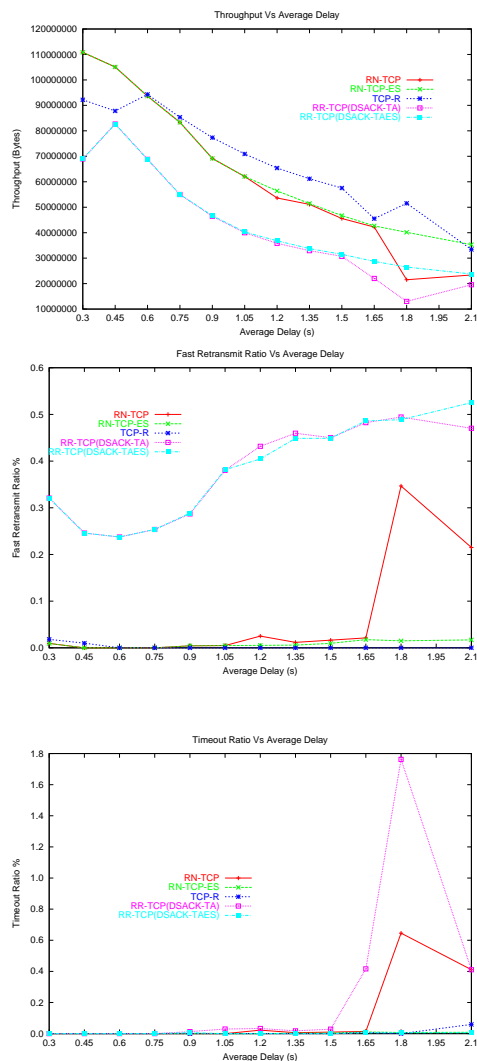


Figure 11: Propagation delay of 300 ms.

the other protocols. Even though RN-TCP-ES has a higher fast retransmit ratio compared to DSACK-TAES, RN-TCP-ES has a lower timeout ratio compared to DSACK-TAES leading to a better throughput improvement.

10.2 Throughput: Large Reordering Delays (Scenario 2)

In this section, we analyze the throughput performance of TCP-R over RN-TCP, RN-TCP-ES, DSACK-TA and DSACK-TAES when packets experience reordering due to link level retransmissions and no packets get dropped due to corruption. The packet delay rate was fixed at 7%. We varied the value of γ from 1.0 to 7.0.

It can be seen from the Figure 11, that when the average packet delay is less than 1.0 s, the throughput performance of RN-TCP-ES and DSACK-TAES is similar to RN-TCP and DSACK-TA. When the average packet delay is more than 1.0 s, the throughput of RN-TCP and DSACK-TA drop rapidly whereas the throughput of RN-TCP-ES and

DSACK-TAES is steady. It is evident from the timeout ratio graph, that both RN-TCP and DSACK-TA have a large timeout ratio unlike RN-TCP-ES and DSACK-TAES which have timeout ratio that hovers around zero. TCP-R performs much better compared to the other protocols. For example, when the average packet delay is 0.9 s, TCP-R gives a 12% throughput performance over RN-TCP and RN-TCP-ES. Moreover TCP-R gives a staggering 65% improvement over RR-TCP(both DSACK-TA and DSACK-TAES). When the average packet delay is 1.65 s, TCP-R gives a 7% throughput improvement over RN-TCP-ES, 8% throughput improvement over RN-TCP, 58% throughput improvement over DSACK-TAES and almost a two fold improvement over DSACK-TA. TCP-R has a zero fast retransmit ratio and almost a zero timeout ratio.

11. CONCLUSIONS AND FUTURE WORK

In this section, we proposed a solution that allows the TCP sender to distinguish whether a packet has been lost or reordered in the satellite network and perform actions accordingly. This was done by maintaining information about dropped packets (only due to congestion in the network) in the gateway and using this information to notify the sender, whether the packet has been dropped or reordered/corrupted in the network. We termed this mechanism as Explicit Packet Drop Notification (EPDNv3.0). We also proposed an extension to SACK protocol called Robust TCP (TCP-R). If the TCP-R assumes the packets to be reordered or corrupted in the network, it immediately retransmits the packet after receiving three duplicate ACKs and enters our modified fast recovery mechanism where the procedure of reducing the *ssthresh* and the *cwnd* are bypassed i.e. we do not reduce the *ssthresh* and the *cwnd*. We also compared TCP-R with other protocols namely SACK, DSACK-R, RR-TCP and RN-TCP. We believe the gateway could be modified to send the dropped information in an ICMP message to the sender. This requires further study and testing. Further simulations and testing needs to be carried out to find the efficiency of the protocol when there is an incremental deployment i.e. when there are some routers in a network which have not been upgraded to use our mechanism. Moreover, the simulated results presented in this paper needs verification in the real satellite network.

12. REFERENCES

- [1] I.F. Akyildiz, G. Morabito, and S. Palazzo, TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks, IEEE/ACM Transactions on Networking, Volume 9, Number 3, Pages 307-321, 2001.
- [2] A. Bakre and B. R. Badrinath, I-TCP: Indirect TCP for Mobile Hosts. Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS), 1995.
- [3] H. Balakrishnan and R.H. Katz, Explicit Loss Notification and Wireless Web Performance, Proceedings of the IEEE Globecom Internet Mini-Conference, Sydney, Australia.
- [4] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM

Transactions on Networking (TON) archive Volume 5 , Issue 6, pp: 756 - 769, December 1997.

- [5] E.Blanton, M.Allman, On Making TCP More Robust to Packet Reordering. ACM Computer Communication Review, 32(1), 2002.
- [6] B. Chinoy, ACM SIGCOMM Computer Communication Review, Volume 23, Issue 4, New York, 45 - 52, 1993.
- [7] J. V. Evans, The US proposed new multimedia communications satellite systems, Proceedings of 2000 IEEE Aerospace Conference, Big Sky, Montana, 18-25, 2000.
- [8] S.Floyd, J.Mahdavi, M.Mathis, M.Podolsky, An Extension to the Selective Acknowledgement (SACK) Option for TCP, RFC 2883, 2000.
- [9] T. R. Henderson, R.H. Katz, Transport Protocols for Internet-Compatible Satellite Networks, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 2, pp. 345-359, February 1999.
- [10] V.Jacobson, Symposium proceedings on Communications architectures and protocols, California, 314 - 329, 1988.
- [11] R. Krishnan, P.G. Sterbenz, W. M. Eddy, C. Partridge, M. Allman, Explicit Transport Error Notification for Error-Prone Wireless and Satellite Networks, Computer Networks journal, Elsevier, 2004.
- [12] R. J. Leopold, Low-earth orbit global cellular communications network, Proceedings of ICC '91, pp. 1108-1111, 1991.
- [13] G. Maral, VSAT Networks, J. Wiley and Sons, 1995.
- [14] S.McCanne, S.Floyd, Network Simulator, <http://www.isi.edu/nsnam/ns/>
- [15] B.Pattan, Satellite-Based Global Cellular Communications, McGraw-Hill, 1997.
- [16] J.Postel, Transmission Control Protocol, RFC 793, 1981.
- [17] A.Sathiaseelan, T. Radzik, Improving the Performance of TCP in the Case of Packet Reordering, Proceedings of the 7th IEEE International Conference on High Speed Networks and Multimedia Communications HSNMC'04, Toulouse, France, 2004.
- [18] C. Ward, H. Choi, and T. Hain, A data link control protocol for LEO satellite networks providing a reliable datagram service, IEEE/ACM Transactions on Networking, 3(1):91103, Feb. 1995.
- [19] L. Wood, G. Pavlou and B. G. Evans, Effects on TCP of routing strategies in satellite constellations, IEEE Communications Magazine, special issue on Satellite-Based Internet Technology and Services, vol. 39 no. 3, pp. 172-181, 2001.
- [20] M.Zhang, B.Karp, S.Floyd, L.Peterson, RR-TCP: A Reordering-Robust TCP with DSACK. 11th IEEE International Conference on Network Protocols (ICNP'03), Georgia, 2003.

APPENDIX

A. TCP-R: RECEIVER SIDE IMPLEMENTATION DETAILS

The TCP-R receiver maintains two lists: the reorder list and the drop list. The elements of these lists are packet sequence numbers. The data packets that arrive at the receiver could bring in the maximum-minimum dropped information about any dropped packets irrespective of the sequence. For example, a packet with lesser sequence number could bring in higher minimum and maximum dropped sequence numbers for that particular flow. The TCP-R receiver has to consider all possible cases before considering whether the gaps caused are due to reordering/corruption or dropped packets.

When a data packet P arrives at the TCP-R receiver, the following computation is done. The TCP-R receiver checks whether $P:n$ is present in the drop list or the reorder/corrupt list. If present, then the entry is deleted from the list (This means that the packet $P:n$ has arrived at the receiver and any entry with the sequence number $P:n$ in any of these lists should be removed) . The TCP-R receiver checks whether the dropped entries $P:min$ and $P:max$ are empty or not.

If the dropped entries $P:min$ and $P:max$ are NULL, the TCP-R receiver checks if $P:n$ is greater than the highest received packet $Q:n$ in the receiver buffer queue.

- If $P:n$ is greater than $Q:n$, then the TCP-R receiver checks for a gap between $P:n$ and $Q:n$ and if those sequence numbers required to fill the gap are present in the drop list.
 - If some of these numbers are in the drop list, then the TCP-R receiver assumes that those packets have been dropped.
 - If not, then the packets within the gap are assumed to be reordered. The TCP-R receiver adds those sequence numbers required to fill the gap to the reorder/corrupt list.

If $P:min$ and $P:max$ are not NULL AND if $P:n > Q:n$.

- The TCP-R receiver checks if $P:min > Q:n$ AND $P:max < P:n$.

The TCP-R receiver checks for a gap between $P:min$ and $Q:n$ and also checks for a gap between $P:max$ and $P:n$. If there is a gap, the TCP-R receiver adds those sequence numbers required to fill the gap to the reorder/corrupt list. Whilst adding, check if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list. Add the sequence numbers from $P:min$ to $P:max$ into the drop list.
- If $P:min > Q:n$ AND $P:max > P:n$ AND $P:min < P:n$.

The TCP-R receiver checks for a gap between $P:min$ and $Q:n$. If there is a gap, the TCP-R receiver adds those sequence numbers required to fill the gap to the reorder/corrupt list. Then the TCP-R receiver checks if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them

from the reorder/corrupt list. Put the sequence numbers from $P:min$ to $P:n-1$ and $P:n+1$ to $P:max$ into the drop list for future references.

- If $P:min > Q:n$ AND $P:min > P:n$.
Add sequence numbers from $P:min$ to $P:max$ into the drop list. Add sequence numbers from $Q:n+1$ to $P:n-1$ into the reorder/corrupt list.
- If $P:min < Q:n$ AND $P:max < P:n$ AND $P:max > Q:n$.
The TCP-R receiver checks for a gap between $P:max$ and $P:n$. If there is a gap, the TCP-R receiver adds those sequence numbers required to fill the gap to the reorder/corrupt list. Then the TCP-R receiver checks if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list. Add sequence numbers from $P:min$ to $Q:n-1$ and from $Q:n+1$ to $P:max$ into the drop list.
- If $P:min < Q:n$ AND $P:max < P:n$ AND $P:max < Q:n$.
Add sequence numbers from $P:min$ to $P:max$ into the drop list. Add sequence numbers from $Q:n+1$ to $P:n-1$ into the reorder/corrupt list.
- If $P:min < Q:n$ AND $P:max > P:n$.
The TCP-R receiver checks if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list. Put the sequence numbers from $P:min$ to $Q:n-1$, $Q:n+1$ to $P:n-1$ and $P:n+1$ to $P:max$ into the drop list for future references.

If $P:min$ and $P:max$ are not NULL AND if $P:n < Q:n$.

- If $P:min > Q:n$.
The TCP-R receiver checks for a gap between $P:min$ and $Q:n$. If there is a gap, the TCP-R receiver adds those sequence numbers required to fill the gap to the reorder/corrupt list. Put the sequence numbers from $P:min$ to $P:max$ into the drop list for future references.
- If $P:min < P:n$ AND $P:max > P:n$ AND $P:max > Q:n$.
The TCP-R receiver checks if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list. Put the sequence numbers from $P:min$ to $P:n-1$, $P:n+1$ to $Q:n-1$ and $Q:n+1$ to $P:max$ into the drop list for future references.
- If $P:max < Q:n$ AND $P:max > P:n$ AND $P:min < P:n$.
The TCP-R receiver checks if the sequence numbers from $P:min$ to $Q:n$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list. Put the sequence numbers from $P:min$ to $P:n-1$ and $P:n+1$ to $P:max$ into the drop list for future references.
- If $Q:n > P:max$ AND $P:min > P:n$.
Add sequence numbers from $P:min$ to $P:max$ into the drop list.
- If $P:min < Q:n$ AND $P:max < P:n$.
The TCP-R receiver checks for a gap between $P:max$ and $P:n$. If there is a gap, the TCP-R receiver adds

those sequence numbers required to fill the gap to the reorder/corrupt list. The TCP-R receiver checks if the sequence numbers from $P:min$ to $P:max$ are present in the reorder/corrupt list. If present, remove them from the reorder/corrupt list and add them to the drop list.