A Sender-based TFRC for *Saratoga*: A Rate Control Mechanism for a Space-Friendly Transfer Protocol

Abu Zafar M. Shahriar and Mohammed Atiquzzaman
Computer Science
University of Oklahoma
Norman, OK 73019
shahriar, atiq@ou.edu

William D. Ivancic
Satellite Networks and Architectures Branch
Communication Technology Division
NASA Glenn Research Center
Cleveland, OH 44135
wivancic@grc.nasa.gov

Lloyd Wood
Centre for Communication Systems Research
University of Surrey
Guildford GU2 7XH, UK
l.wood@surrey.ac.uk

Abstract—Saratoga is a protocol for fast file transfers across dedicated links in private networks, using small amounts of feedback for loss recovery. It is in use to download large amounts of imaging data from remote-sensing satellites, where the link environment is highly asymmetric and uplinks are constrained. However, *Saratoga* lacks a rate-control mechanism to allow fair share with co-existing flows for simultaneous competing transfers, or for across the congested Internet where it must coexist fairly with TCP. TFRC, a self- and TCP-Friendly Rate Control mechanism, can be adopted for *Saratoga* and leverage its existing protocol information. Use of TFRC normally requires significant changes in protocol operation, including additional data in feedback. We design a sender-based TFRC for *Saratoga*, needing only simple modifications within the sender and using only existing feedback information. This sender-based TFRC is shown to share the bottleneck-bandwidth fairly under various network conditions, allowing *Saratoga* to be adapted for shared links or for the congested Internet, while still supporting the asymmetric environments that *Saratoga* was originally developed for.

TABLE OF CONTENTS

1 INTRODUCTION 1
2 RECEIVER-BASED TFRC 3
3 SENDER-BASED TFRC (STFRC) FOR *Saratoga* .. 3
4 PERFORMANCE EVALUATION 6
5 DISCUSSION 9
6 CONCLUSION 10
ACKNOWLEDGEMENTS 10
REFERENCES 11
BIOGRAPHY 11

1. INTRODUCTION

Wood *et al.* [1] describe *Saratoga*, a UDP-based protocol that sends data at a rate independent of the rate of feedback, and performs loss recovery based on periodic feedback. *Saratoga*'s loss recovery mechanism is less chatty than others' loss recovery mechanisms to support constrained return-channels. *Saratoga* is suitable for the use in satellite-environments where links are highly asymmetric having brief periods of connectivity, and where loss occurs due to channel-related errors, rather than to congestion. Uplinks are constrained in such environments.

Currently, *Saratoga* is being used to download data from Internet Protocol (IP)-enabled Disaster Monitoring Constellation (DMC) satellites constructed by Surrey Satellite Technology. The five DMC satellites currently operational in low Earth orbit, of the seven that have been launched provide remote-sensing images to support disaster relief. Remote-sensing images provided by these DMC satellites are used for the observation of the Earth to monitor flood, wildfires, volcanoes and cryosphere events, as well as agricultural and population monitoring. The usefulness of DMC satellites for the observation of the Earth has been evaluated in [2].

Saratoga has also been used to demonstrate delay-tolerant networking concepts from orbit, with the first tests of the Bundle Protocol from space for the Interplanetary Internet to deliver images as bundles [3]. This is an optional capability provided by *Saratoga*, and not in regular operational use. *Saratoga* is also currently being evaluated for use in private radio astronomy networks, where high-speed sensor data flows are a base requirement [4].

Saratoga could be used to download data simultaneously from multiple IP-enabled devices onboard satellites [5], rather than using the scheduled one-file-only-after-another model that currently avoids competition, or could be used for transfers of data from remote-sensing systems directly to end

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC Paper #1444, Version 2, Updated 09/01/2011.

users through the public Internet [6]. Such data transfers will be across links which are shared by co-existing flows or with other protocols, predominantly Transmission Control Protocol (TCP), in the Internet.

TCP is the most widely-used reliable and rate-controlled transport protocol when multiple competing flows share common links. TCP achieves reliability and rate-control based on frequent acknowledgments that can be a limiting factor for data transfers when the forward/back-path-asymmetry exceeds 50:1 [7]. That, and TCP's assumptions about all packet loss being caused by congestion in link buffers and queues, make TCP unsuitable for the satellite environment we have described, as a single TCP flow will not be able to fully utilize the available capacity of satellite links.

An optional TCP-friendly rate-control mechanism is desirable in *Saratoga* to permit fair allocation of shared paths to TCP and other traffic, and to enable *Saratoga* to be used in the public Internet, rather than only in the private networks for which *Saratoga* was originally designed and developed. When we refer to rate control here, we mean a closed feedback loop leading to managed flow control, rather than controlling to a fixed rate in an open loop.

We aim to design a TCP-friendly rate-control mechanism for *Saratoga*. Widmer *et al.* [8] provide an overview of known TCP-friendly rate-control mechanisms. Given *Saratoga*'s data sending mechanism, we prefer the rate-based approach to the window-based one for better integration and to keep the functionality of the *Saratoga* sender simple. More importantly, considering the current use of *Saratoga* in private space links where losses can be bursty and are due to channel-related errors, not to congestion, the conservative reaction of rate-based approaches to packet loss will provide better throughput performance than the more aggressive reaction of window-based approaches. Among the rate-based protocols proposed in the literature, TCP Friendly Rate Control (TFRC) [9, 10] suits *Saratoga* in terms of low needed feedback rate. TFRC, which is specified by the Internet Engineering Task Force as a proposed standard [9], has been shown to perform very well for a variety of available link capacities and number of flows [10]. Therefore, we *aim* to use a TFRC-like rate-control mechanism with minimal changes to the existing *Saratoga* protocol.

In TFRC, the sender controls the rate of sending data using a model imitating the long-run behavior of TCP, and requires two parameters from the receiver to do this – a measure of the loss and the receiver's throughput. It is *receiver-based* because the parameters are computed at the receiver by keeping a history of receive-times of packets, and are sent to the sender through periodic feedback. Although the computations could be done at the sender in the sender-based variants discussed in [9], the receiver still needs to send feedback containing the receiver's throughput, loss-related information and other components required to compute the measure of loss. Moreover, the Round Trip Time (RTT), which is required at the receiver in the above-mentioned variants of TFRC, has to be either sent from the sender to the receiver or sampled at the receiver. However, sampling the RTT at the receiver has ill-effects on the performance of TFRC suggesting to send the RTT from the sender to the receiver [11]. The sending of the RTT from the sender, and the above-mentioned feedback is not supported in *Saratoga* protocol.

A *sender*-based TFRC has been proposed in [12], where the measure of loss is computed at the sender by keeping a history of send-times of packets. To compensate the error due to the use of send-times instead of receive-times, the measure of loss is corrected by the ratio of the receiver's throughput to the sending rate. Like the receiver in *Saratoga*, the receiver in this sender-based version of TFRC sends the loss report to the sender through feedback. However, feedback packets, which contain bit-fields indicating the fate (lost or received) of packets, differ from feedback packets in *Saratoga* where only the offsets of lost packets are sent. Moreover, unlike the receiver in *Saratoga*, the receiver in the sender-based version of TFRC sends the receiver's throughput to the sender.

The previous two paragraphs suggest that adopting existing TFRC mechanisms would require significant modifications to the *Saratoga* protocol. Moreover, sending additional data, required for existing TFRC versions, in feedback is undesirable in *Saratoga* when asymmetry and low rates of the return-channel are present, as acknowledgement congestion on the return-channel is a concern. Therefore, we *design* a true sender-based TFRC-like mechanism that controls the rate using the existing feedback specified in *Saratoga*, which anticipates the use of TFRC or a similar mechanism [1]. Our mechanism resembles the receiver-based TFRC, but computes parameters at the sender.

Our *contributions* are:

1. a rate-control mechanism for *Saratoga*,
2. a sender-based TFRC for Saratoga without significant changes to the existing protocol, incurring no additional feedback, and
3. a major step towards a true sender-based TFRC.

Our results suggest that the sender-based TFRC is self- and TCP-friendly across both symmetric and asymmetric link environments. Results also show that the change of the rate in steady state is smooth, indicating a less aggressive response to the loss and small throughput reduction when the loss is not due to the congestion.

Simple modifications to the sender ensure minimal development effort, while receivers can still receive data from existing senders. True sender-based TFRC shifts the processing and resource requirements to the sender, and therefore, may increase the suitability of TFRC for some servers receiving files from many sources.

The rest of this paper is organized as follows. An overview of the receiver-based TFRC mechanism is presented in Sec. 2. Sec. 3 presents the sender-based TFRC followed by evaluation results in Sec. 4. Sec. 5 highlights conditions for the effective use of the sender-based TFRC followed by concluding remarks in Sec. 6.

2. Receiver-based TFRC

TFRC is a rate-control mechanism with a smoother throughput than TCP, while sharing the bandwidth fairly with TCP [9]. Following are the advantages of TFRC over TCP:

- Decreased variation in the instantaneous throughput. Particularly, TFRC does not cut the throughput to half (fast recovery) when a loss is detected.
- Rate-based mechanism is suitable for rate-based transfer protocols. In *Saratoga*, the rate of sending packets is determined from the data rate which can be obtained directly from the TFRC like mechanism.
- Requires less frequent feedback from the receiver than what is required in TCP.

The basic underlying principles of TFRC, and functionalities in TFRC receivers and senders are now given.

a) Basic principle

TFRC uses the following model [9] which is a simplified form of the model of TCP [13] to compute the data rate (X) as a function of the packet size (s), the RTT (R), a notion of loss (p), an approximation of TCP timeout value (t_{RTO}), and the maximum number of packets acknowledged by a TCP-acknowledgement (b):

$$X = \frac{s}{R \times \sqrt{2bp/3} + t_{RTO} \times \left(3 \times \sqrt{3bp/8}\right) \times p \times (1 + 32p^2)} \quad (1)$$

The value of p is computed and sent by the receiver to the sender. For the computation of the value of p, losses separated by a time period of an RTT or more are recognized as loss events. The value of p is computed as the reciprocal of the weighted average of the number of packets sent between the start of two successive loss events.

b) Receiver functionalities

This TFRC is *receiver-based* because the receiver computes the value of p and the throughput to provide feedback to the sender, as follows:

1. At reception of a data packet, the receiver records the reception time. If a loss of packets is detected, the supposed reception times of lost packets are interpolated using the times and sequence numbers of packets received right before and after the loss. These times are used to update a *history of lost/received packets*. The value of p is computed from the history. If the value of p has increased compared to the last computed value, the receiver's throughput is computed, and this information is sent as feedback to the sender.
2. The receiver should send a feedback packet at least every RTT if data packets have been received since the last feedback packet was sent. The value of p and the receiver's throughput are computed using the history of packets, and are included in the feedback packet.

c) Sender functionalities

1. The sender starts sending data with an initial rate.
2. The sender tracks the weighted average of the RTT and an approximation of the TCP-timeout value. When a feedback packet is received from the receiver, the sender updates the weighted average of the RTT, and approximates the TCP-timeout using the RTT. The value of p and the receiver's throughput contained in the feedback packet are used to update the sending rate. If the value of p is zero, the sender doubles the current rate, which is bounded by twice the receiver's throughput at the higher side, and one packet every RTT at the lower side. Otherwise, the sender computes the value of X as the sending rate, which is bounded by twice the receiver's throughput at the higher side and one packet every 64 seconds at the lower side.
3. If no feedback packet is received for a certain period of time, the sender cuts down the rate to half.

3. Sender-based TFRC (STFRC) for *Saratoga*

In this section, we present the challenges in designing the Sender-based TFRC (STFRC) for *Saratoga*, our approaches to meet those challenges, and the algorithms used for the proposed STFRC.

a) Overview of our approach

In STFRC for *Saratoga*, the receiver is unchanged from *Saratoga*. The sender performs all rate-control related functionalities of TFRC. These functionalities include building the history of packets followed by the computation of the receiver's throughput, the loss event rate and the sending rate. Building the history of packets requires reception-times and packet-delivery fates. Since the sender can only learn the loss status of packets when a feedback packet (so-called STATUS-feedback packet) is received, the updating of the history and computations are performed at reception of a STATUS-feedback packet. Reception-times are predicted from send-times of packets and the RTT. Therefore, the sender records the send-time of a packet in the history at the time of sending, and adds a fraction of RTT to the send-time when the STATUS-feedback packet is received. Also, the packet-delivery-fates are marked in the history using the report of loss contained in the STATUS-feedback packet. After updating the history, the computations are performed and the newly-computed sending rate is used till the next STATUS-feedback packet is received. Given Saratoga's feedback mechanism, implementation of the functionalities raises some challenges in designing the STFRC for *Saratoga*.

We present these challenges, our approaches to meet the challenges, and the algorithms used for the proposed STFRC in the following subsections.

b) Challenges to design the STFRC for Saratoga

Building the history of packet delivery requires the reception-times of packets. Lost and retransmitted (or delayed and retransmitted) instances of the same packet must be uniquely identified in the history for the accurate computation of the average receiver's throughput over the last RTT, and for the computation of the value of p.

These requirements give rise to the following challenges to design the STFRC for *Saratoga*:

• Determining the reception-times: As packet reception-times are neither known to the sender nor sent from the receiver, the sender has to predict the reception-times as correctly as possible without incurring a significant overhead.
• Unique identification of packets: In *Saratoga*, the receiver sends offsets of data in STATUS-feedback packets to report lost and received packets. These offsets are not unique or sequential, due to losses and retransmissions. The same losses may be reported in multiple STATUS-feedback packets if multiple requests are received before receiving the lost packets reported in the first of the STATUS-feedback packets. Therefore, a mechanism is needed to uniquely identify packets without incurring too much overhead.

c) Our approaches to meet the challenges

We address the above-mentioned challenges as follows:

I. Prediction of reception-times of packets—An estimation of the forward-path-delay is obtained by multiplying the RTT with a *Symmetry Ratio*, defined as the ratio of the average forward-path-delay to the average RTT. Since the size of packets on the forward path is larger than that on the reverse path, the *Symmetry Ratio* may not be equal to 0.5 because of the larger transmission delay and larger probability of developing congestion at local exit routers (considering equal delay at intermediate routers). Therefore, it would be better to obtain the factor from the long-term knowledge of the network or using a low-overhead mechanism to estimate the forward delay periodically at the cost of increased overhead. Reception-times are obtained by adding the forward delay to the send-times of packets. The RTT can be measured periodically using timestamps in the STATUS-feedback packet. (Timestamps are optional in *Saratoga*.)

II. Unique identification of packets—Unique identification is required for a history of packet-delivery fates that can be updated by the sender when a STATUS-feedback packet is received. Although the STATUS-feedback packet contains a report up until the packet requesting the STATUS-feedback packet to be sent, the history may contain send-times of packets sent after sending the packet carrying the request. This happens because the sender will continue to record the send-times of packets that will be sent between the time of sending the packet carrying the request and the time of receiving the STATUS-feedback packet that was requested. When the STATUS-feedback packet is received, the sender needs to move back along the history from the time of receiving the STATUS-feedback packet to the time of receiving the packet that sent the request for the STATUS-feedback packet. This is done to confine the updating only to those packets whose receive-times and delivery-fates can be determined from this particular STATUS-feedback packet. Therefore, the history is updated by determining the receive-times and delivery-fates of the packets sent during the time period between sending two successive successfully-answered requests for STATUS-feedback packets.

To better explain which packets in the history are updated at reception of a STATUS-feedback packet, we introduce Fig. 1 demonstrating various events that may happen during an ongoing transmission in *Saratoga*. While the data transmission is going on, and a request for STATUS-feedback packet is due, the sender sends a request with the very next packet sent at time $t1$. At reception of the packet with the request, the receiver responds with a STATUS-feedback packet containing a report of losses of packets received so far. Assuming packets arrive at the receiver in the sequence they are sent, this STATUS-feedback packet received at time $t1'$ will contain report up until the packet sent at time $t1$. The sender continues to send packets and record send-times in the history after sending the packet at time $t1$. Thus, when the STATUS-feedback packet is received, the history will contain packets up until the packet sent at time $t1'$. However, the conversion of send-times to receive-times and marking delivery-fates of packets are performed for those packets (whose send-times are in the history) that were sent up until the packet sent at time $t1$.

The next request is sent at time $t2$ and is lost on the way (alternatively, the corresponding STATUS-feedback packet might get lost). We reckon the time (e.g. from $t1$ to $t2$ or from $t2$ to $t3$) between sending two successive requests as a *period*

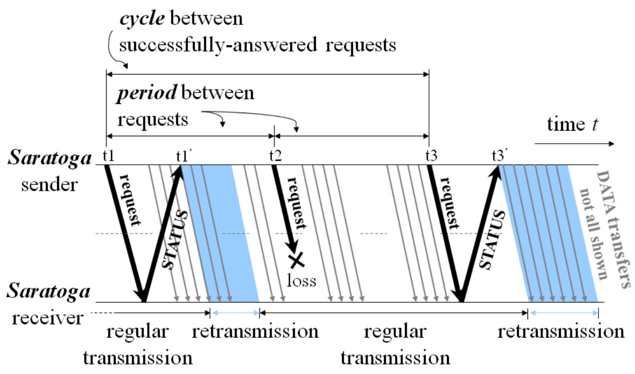


Figure 1. Ladder diagram showing various transmission events in *Saratoga*.

which is significant because various information (presented later in the section) regarding some packets, sent during the *period*, is recorded. At time $t3$, the sender sends the next request which is answered successfully by the receiver. At reception of this STATUS-feedback packet at time $t3'$, the sender updates the history of packets that were sent between time $t1$ and $t3$. We call this time period, $(t3 - t1)$, a *cycle*. A *cycle* may consist of one or more *periods*. *Saratoga* has an explicit but optional timestamp field which is used to identify the *cycle*.

The identification of packets sent during a *cycle* is required for updating the history. Unique identification of packets sent during a *cycle* would be possible by storing their offsets and send-times. But this is inefficient due to the requirement for large amounts of memory to store offsets and additional processing to identify packets using the send-times. Inefficiency increases when the number of packets sent during a *cycle* is large due to the high sending rate, and/or the loss of request/STATUS-feedback packets resulting in a long *cycle*. Therefore, we use an alternative method for identification of packets.

For the identification of packets using the alternative method, we introduce dummy sequence numbers used only within the sender. When a packet is sent, the send-time is recorded in the history, and the packet is assigned a unique dummy sequence number which is used to identify the packet in the history. At reception of a STATUS-feedback packet, the sender determines the range of dummy sequences of the packets sent during the *cycle* ended by this STATUS-feedback packet. A range determination is required because the history might contain packets sent in previous and subsequent *cycle*s. Dummy sequences of lost packets are also determined from the offsets of lost data reported in the STATUS-feedback packet, and from the information described below. The range of dummy sequences, and the dummy sequences of lost packets are used to identify the packets in the history to convert their send-times to receive-times by adding the predicted forward delay, and to mark them as lost/received.

For the identification using dummy sequences, we recognize the following sets of transmissions, shown in Fig. 1, that might happen during a *period*:

- Set I of regular transmission: First set of regular transmission starts with the packet sent after the packet carrying a request (e.g. sent at time $t1$), and occurs until the reception of a STATUS-feedback packet. If the STATUS-feedback packet does not report any loss, then this set of transmission continues until sending the next request (e.g. up until the packet sent at time $t2$).
- Set of retransmissions: Retransmissions (shown in dark background in Fig. 1) start after receiving a STATUS-feedback packet containing reports of losses, and continues until all lost data are retransmitted. The set of retransmissions contains all the retransmissions that occur in the *period*. Retransmissions may not occur if no loss is reported.
- Set II of regular transmission: It starts after the end of retransmissions (if occurs), and continues until sending the next request (e.g. up until the packet sent at time $t2$).

The following information is recorded for identification purposes:

- Dummy sequences and send-times of the first and the last packet of each period: These are required to identify the packets sent during a cycle, and to determine the span of a cycle.
- Offset to dummy sequence mapping for all retransmitted packets: Since offsets of retransmitted packets are not sequential, these are required to find the dummy sequences of packets lost from retransmitted packets.
- Offsets and dummy sequences of the first packets of each set of regular transmissions: Since offsets of regular packets are sequential, dummy sequences of packets that are lost from regular transmissions can be obtained from offsets of lost packets using the first packet's dummy sequence and offset, and the packet size.

The information is recorded for each *period* because a *period* is a potential *cycle*. If two STATUS-feedback packets sent in response to two successive requests marking a *period* are received, the *period* is a *cycle*. If a packet carrying a request or the corresponding STATUS-feedback packet is lost, the *cycle* consists of more than one *period*. When a STATUS-feedback packet is received, the sender uses the timestamp in the packet and the time of the last packet sent in each of the periods to determine whether the *cycle* consists of multiple *period*s or not, and the information for those *period*s are merged for the *cycle*.

d) Sender algorithms

The steps that are executed by the sender to implement the approaches mentioned in this section are shown in Algorithm 1 and 2, and are discussed below.

Algorithm 1 Sender's algorithm when a packet is sent.

```
1: Detect the transmission set type, and record sequence
   numbers, offsets and send-times of packets as discussed
   in Sec. 3.
2: if (a request for a STATUS-feedback packet is due) then
3:     Record the dummy sequence and the send-time of the
       packet to mark the end of a period.
4: end if
5: if (a request was sent with the previous packet) then
6:     Record the dummy sequence and the send-time of the
       packet to mark the start of a period.
7: end if
8: Store the send-time of the packet in the history.
```

The steps given in Algorithm 1 are required for unique identification of packets, and for congestion control. These steps are in addition to the steps that are executed by a *Saratoga* sender without congestion control.

Algorithm. 2 lists the steps that are executed by a sender in STFRC for *Saratoga* when a STATUS-feedback packet is received.

Algorithm 2 Sender's algorithm when a STATUS-feedback packet is received.

```
1: Update the RTT and the TCP-timeout.
2: Identify the cycle i.e. the dummy sequence number of the
   last packet ending the cycle. This might require merging
   of multiple periods into a cycle.
3: Using the recorded information specified in Algorithm 1,
   offsets of the lost packets reported by the receiver, and
   the information from Step 2, find dummy sequences of
   the packets lost in the cycle.
4: Update receive-times and delivery-fates (received or lost)
   of the packets sent during the cycle.
5: Estimate the receiver's throughput and compute the value
   of $p$.
6: Compute the sending rate.
7: Prepare the retransmission list from the loss reported.
```

In Algorithm 2, Step 6 is similar to the computation of the sending rate by a TFRC sender discussed in Sec. 2, whereas Step 7 is for the response of a typical *Saratoga* sender. The requirement for Step 2 is explained next. As discussed earlier in this section, a *cycle* consists of multiple periods when requests or STATUS-feedback packets are lost. This requires merging of multiple *period*s into a *cycle*. The reason for finding the range of dummy sequences of packets sent in a *cycle* has been explained in the 5th paragraph of Sec. 3-c(II).

Step 3 is required for two reasons. First, a STATUS-feedback packet may report losses that have been already reported by previous STATUS-feedback packets because the STATUS-feedback packet was sent before the retransmitted packets have reached the receiver. Therefore, packets lost in the current *cycle* have to be identified. Second, offsets of lost packets have to be mapped to dummy sequences to update the packet-delivery-fates in the history. Identification and mapping have been discussed earlier in this section.

In Step 4, times and delivery-fates of packets are updated in the history based on the cycle identified in Step 2, and the dummy sequences of lost packets obtained in Step 3. In Step 5, the receiver's throughput and the value of p are determined in a similar way it is done in TFRC [9].

4. Performance evaluation

We use *ns*-2 [14] simulations to evaluate the self- and TCP-friendliness of STFRC for *Saratoga*.

a) Simulation environment

We evaluate two simulation topologies: one using symmetric wired links around a bottleneck, and another adding an asymmetric satellite link. The bottleneck environment using symmetric links is similar to that used in [10] for the evaluation of the receiver-based TFRC, and is intended to show that STFRC performs as well as the receiver-based TFRC, under both the conditions for which Saratoga was designed and more general use. The addition of an asymmetric link is intended to demonstrate the performance when capacity of the reverse feedback path is much lower than that of the forward data path.

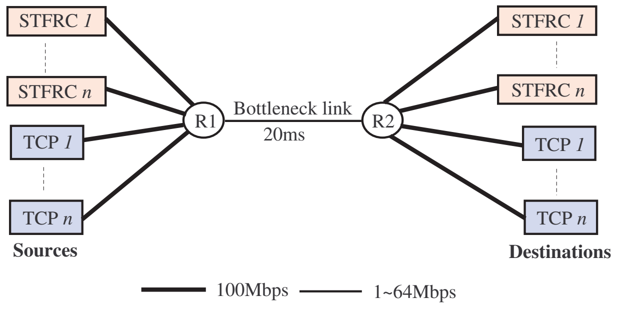


Figure 2. Topology using symmetric wired links.

Fig. 2 shows the topology using symmetric links. An equal number of TCP SACK and STFRC (in *Saratoga*) flows, transferring bulk data, share the bottleneck link. Nodes that contain sources and destinations are connected to routers $R1$ and $R2$, respectively. Queue lengths at routers are scaled according to the bottleneck link bandwidth in a similar way as to [10].

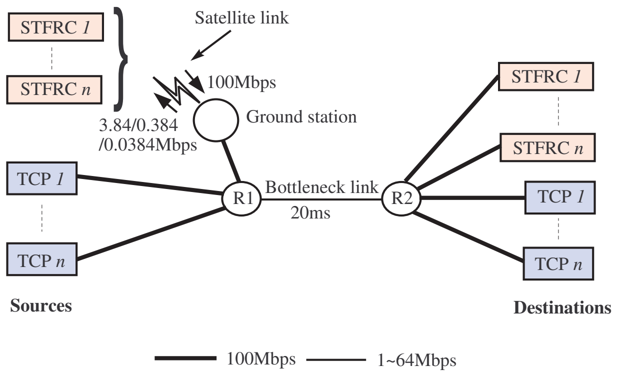


Figure 3. Topology using an asymmetric satellite links.

Fig. 3, based on Fig. 2, adds an asymmetric satellite link. However, instead of using individual nodes connected to $R1$ for each STFRC (in *Saratoga*) source, one satellite node containing all the STFRC sources is connected to a satellite ground station through a wireless link with a relatively high bit error rate of 10^{-4}.

The downlink for the wireless link is simulated at 100Mbps, approximating what is planned for future DMC satellites. We use three uplink bandwidths for the wireless link – 3.84Mbps, 0.384Mbps and 0.0384Mps. An uplink of 0.0384Mbps is planned for deployment in future DMC satellites (where existing DMC satellites currently use 9600bps or 19200bps uplinks). *Unless mentioned explicitly, all results for the asym-*

Table 1. Values of parameters used in the simulation.

Parameter	Value
TCP version	TCP SACK
TCP maximum window size	10000 pkts
Packet size	1040 bytes
Wired link BW (except bottleneck)	100Mbps
Source/destination-R1 wired link delay for the symmetric link case	2ms
Ground station-R1 wired link delay	2ms
TCP source-R1 wired link delay for the asymmetric link case	6ms
TCP/STFRC destination-R2 wired link delay	Variable
Bottleneck wired link delay	20ms
Queue limit at bottleneck link	$BW * 25$
RED queue threshold	$3 + BW * 1.5$
RED queue maximum threshold	$10 + BW * 5$
Satellite wireless downlink	100Mbps
Satellite wireless uplink	Variable
Satellite link error rate	10^{-4}
Symmetry Ratio for the Droptail queue	0.75
Symmetry Ratio for the RED queue	0.65
Smoothing for TFRC	yes
History discounting for TFRC	yes
Simulation time	200sec

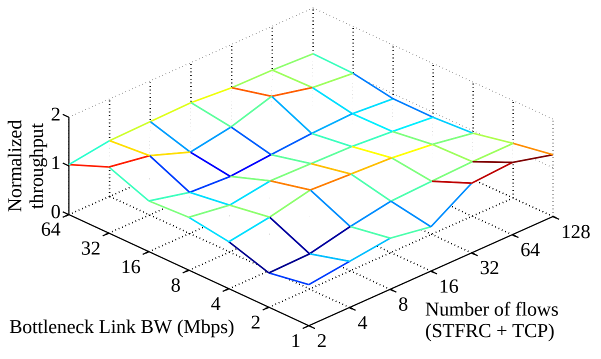
metric link case, presented in this paper, are for the uplink with 0.384Mbps and RED queue at bottleneck links.

Values of the *Symmetry Ratio* used in simulations are 0.75 and 0.65 for Droptail and RED queues, respectively, and are obtained from the ratio of the average forward delay to the average RTT found in the simulation. Values of parameters of STFRC are the default values suggested in [9]. Values of the parameters used in the simulation are summarized in Table 1.

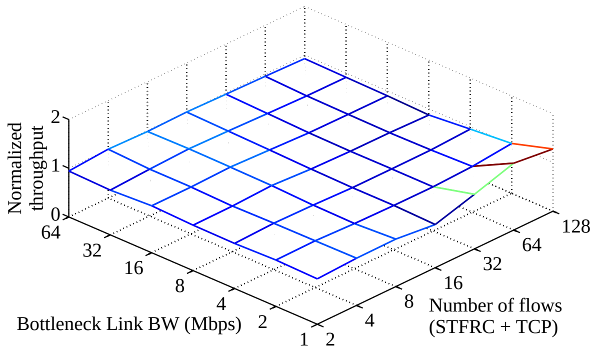
b) Results

To evaluate the fairness to TCP, we measure the throughput of individual flows and the aggregate TCP flows, and variations in individual flows' throughput. To show the response to losses, we measure the instantaneous throughput of all the flows. The results are presented in the following subsections.

I. Normalized throughput of TCP—The normalized throughput of TCP is the aggregate throughput normalized by the share of the bottleneck bandwidth, with a value of one indicating the fair share. Aggregate throughput is measured as the sum of the data received per second at all TCP destinations. Since an equal number of flows of each of STFRC and TCP share the same bottleneck for data packets, and are expected to get an equal share of the bottleneck bandwidth, the normalization is achieved by dividing the aggregate throughput by the half of the bandwidth of the bottleneck link. To simulate a wide variety of network conditions, we vary the number of flows and bottleneck link bandwidth. For the symmetric link case (i.e. for the topology presented in Fig. 2), Fig. 4 shows the normalized throughput for the last 60 seconds of simulation for Droptail (4(a)) and RED (4(b)) queues. Results illustrate the fairness of STFRC over symmetric links.



(a) Droptail queue

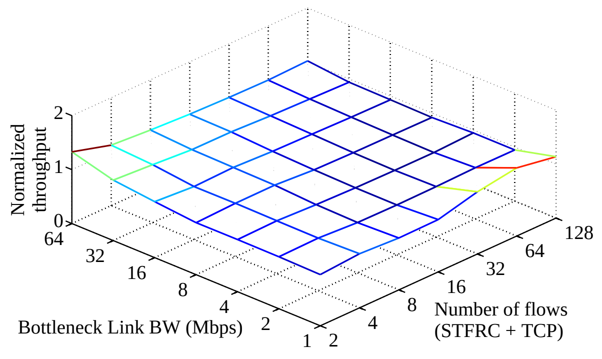


(b) RED queue

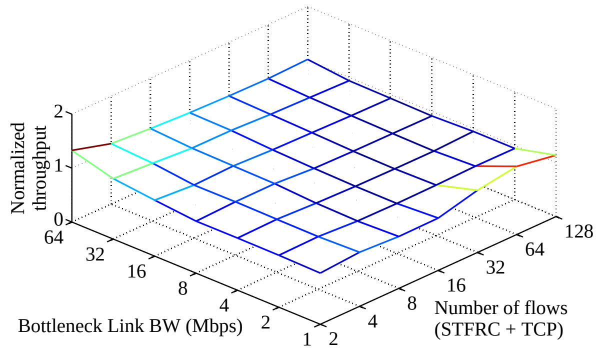
Figure 4. Normalized throughput of TCP for the symmetric link case (topology shown in Fig. 2).

TCP throughput is a little lower than its fair share when the number of flows and the bandwidth are small because TCP is more bursty than STFRC and suffers more drops at small available bandwidth. The TCP throughput is a little higher than that of STFRC at small available bandwidth when the number of flows is large. This happens because flows are forced to operate with a very low window size/rate under such conditions. And even considering TCP's timeouts, we find that while the TCP window size determining the rate cannot go below 1 packet, the rate of an STFRC sender can go below the rate equivalent to that window size.

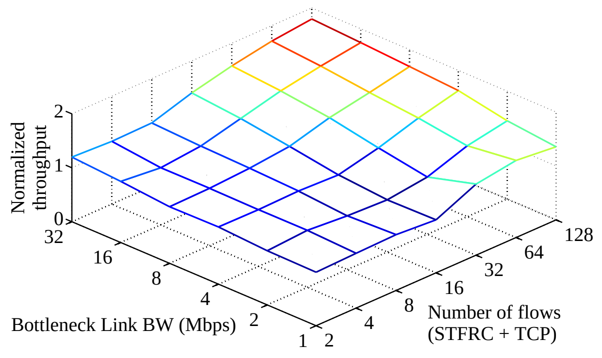
Fig. 5 shows similar results over asymmetric links (i.e. for the topology presented in Fig. 3) for the RED queue used the bottleneck links. Results show the fair share of the bottleneck when the uplink is 3.84Mbps (Fig. 5(a)) and 0.384Mbps (Fig. 5(b)). However, for an uplink of 0.0384Mbps, results (Fig. 5(c)) show the dominance of TCP over STFRC when the bottleneck bandwidth and the number of flows are large. In Fig. 5(c), we present results up to 32Mbps of bottleneck band-



(a) Uplink = 3.84



(b) Uplink = 0.384



(c) Uplink = 0.0384Mbps

Figure 5. Normalized throughput of TCP for the asymmetric link case (topology shown in Fig. 3).

width because STFRC gets a reasonable share up to 16Mbps of bottleneck bandwidth. For the case of 128 flows (64 of TCP and 64 of STFRC) from Fig. 5, a comparative view of the normalized throughput of TCP for various bandwidths of the uplink is presented in Fig. 6 that shows STFRC's failure to have the fair share of the bandwidth. The reason for STFRC not getting the enough share of the bandwidth is the increase of the RTT due to the increase of the End-to-End (E2E) delay on the path from the STFRC destinations to sources.

We measured the E2E delay from STFRC-destinations to

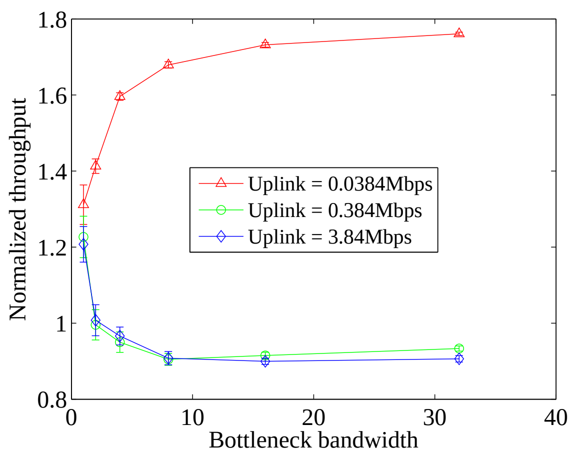
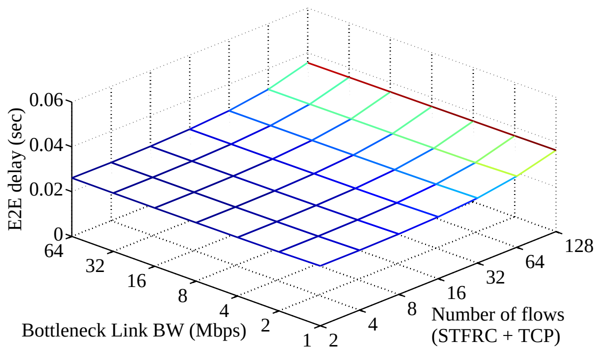


Figure 6. Normalized throughput of TCP when the number of flows is 64 each for the RED queue and the asymmetric link case (topology shown in Fig. 3).

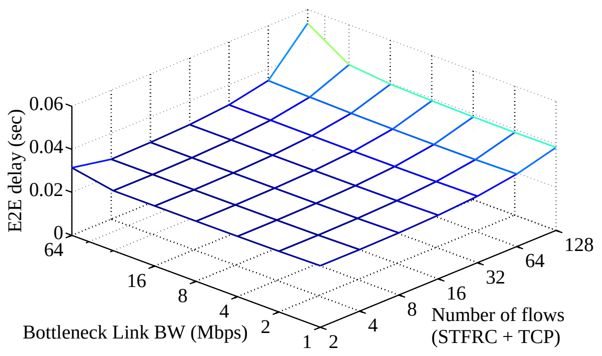
sources. The E2E delay is the difference between the time of sending a STATUS-feedback packet from the destination and the time of receiving the packet at the sender. The E2E delay is shown in Fig. 7 for the three uplink bandwidths. For the uplink bandwidth of 0.0384Mbps, the E2E delay (Fig. 7(c)) increases due to the increase of the rate of feedback beyond the capacity of the uplink when the number of flows increases. The rate of sending feedback in Mbps for the three uplink bandwidths are presented in Fig. 8. The rate of sending feedback in Mbps is measured as the amount of STATUS-feedback packets in Megabytes sent by the receiver per second. As observed in Fig. 8(c) showing the feedback rate for 0.0384Mbps uplink, the feedback rate is around the uplink bandwidth when the number of flows is large. Since the uplink bandwidth of 0.0384Mbps is not sufficient to handle the increased feedback rate, the queuing delay at the uplink increases resulting in an increase in the E2E delay.

II. Normalized Throughput of individual flows—We measure the normalized throughput of individual flows to show the fairness at the flow level. The normalization is performed by dividing a flow's throughput with its expected share of the bottleneck bandwidth. For 16Mbps bottleneck bandwidth and RED queue, Figs. 9 and 10 show the normalized throughput of each flow as well as the mean of those for symmetric and asymmetric links, respectively. Results show that the throughput does not vary wildly around the mean, indicating the overall fairness between individual flows.

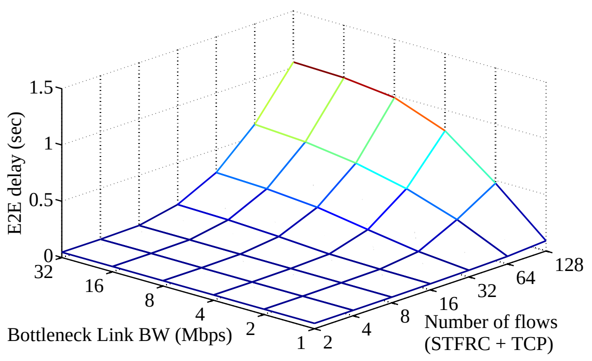
To numerically track the variance of the throughput, we measure the Coefficient of Variance (CoV) of the throughput of individual flows as a function of the loss that influences the CoV. For the symmetric link case, we perform 10 simulation runs for each set of parameter values involving 32 TCP and 32 STFRC flows with varying bottleneck link bandwidths. Fig. 11 shows that the CoV for STFRC is lower than that of TCP until the loss rate reaches 13%. CoVs across the runs



(a) Uplink = 3.84Mbps

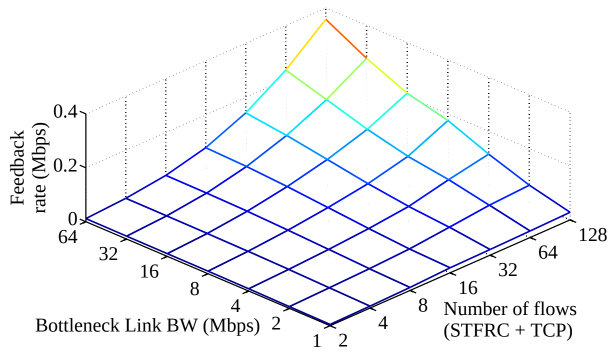


(b) Uplink = 0.384Mbps

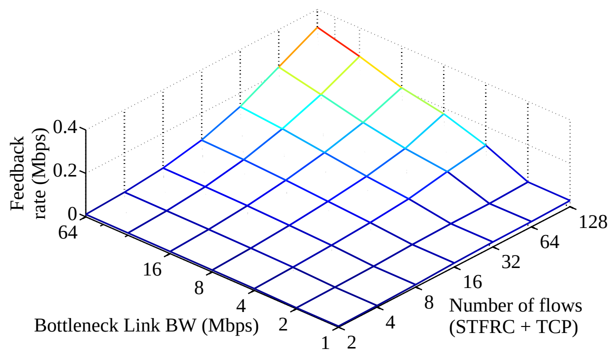


(c) Uplink = 0.0384Mbps

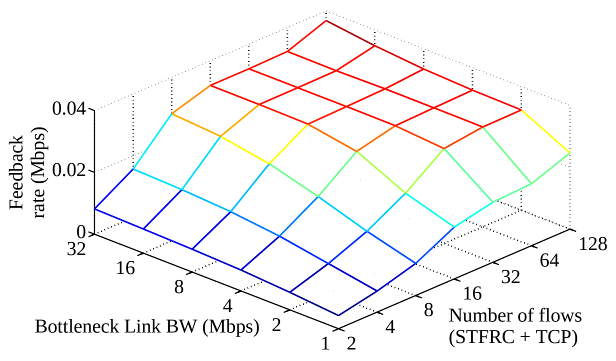
Figure 7. E2E delay from STFRC-sources to -destinations for the asymmetric link case (topology shown in Fig. 3).



(a) Uplink = 3.84Mbps



(b) Uplink = 0.384Mbps



(c) Uplink = 0.0384Mbps

Figure 8. The rate of sending feedback of STFRC-destinations for the asymmetric link case (topology shown in Fig. 3).

also do not vary much. These results illustrate the better inter-flow fairness of STFRC, and conform to the results shown in [10].

III. Sending rate and losses—Fig. 12 shows the sending rate at 0.1 second intervals, and times of losses for one arbitrary flow of STFRC and TCP each. Data is taken from the symmetric link case, where 32 TCP and 32 STFRC flows share a bottleneck bandwidth of 16Mbps. Results show STFRC's less aggressive reduction of the sending rate in response to losses. This is an advantage when bursty losses occur due to reasons other than congestion.

5. Discussion

Numerical results show that STFRC shares bandwidth fairly with co-existing flows and TCP, over both symmetric and asymmetric links. The key factor is the selection of a reasonable value for the *Symmetry Ratio*. We obtain the value from the ratio of the average forward delay and the average RTT found from the simulation. We also measured the nor-

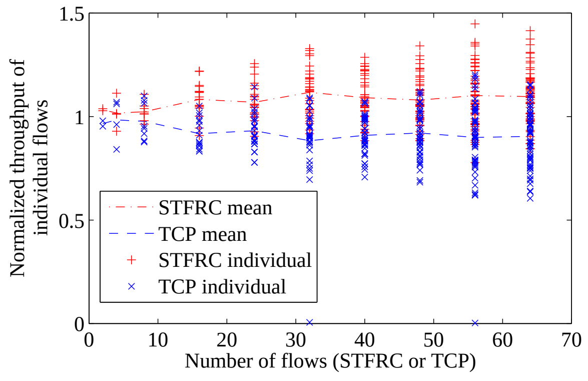


Figure 9. Normalized throughput of individual flows over symmetric links (topology shown in Fig. 2).

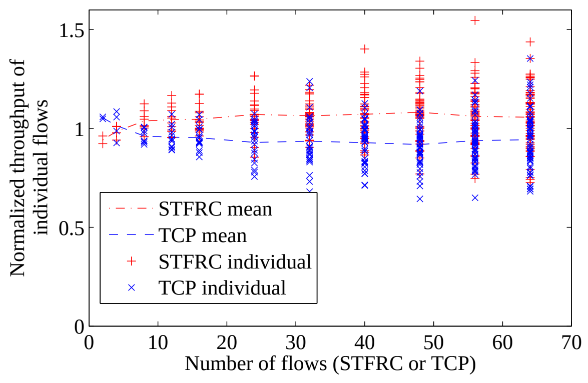


Figure 10. Normalized throughput of individual flows over asymmetric links (topology shown in Fig. 3).

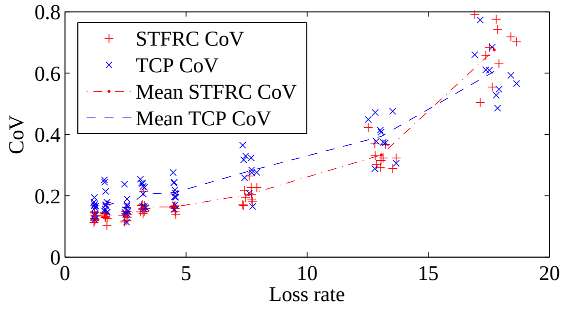


Figure 11. CoV of throughput of flows over symmetric links (topology shown in Fig. 2).

malized throughput of TCP for other values of the *Symmetry Ratio* (e.g. 0.5, 0.6, 0.7) around the obtained value (0.65), and the results were similar, but are not shown because they do not reveal any new findings. When used in a known network environment, this value can be obtained from the knowledge of the network. Otherwise, there are low-overhead estimation methods that can be used to periodically obtain the forward delay. Unless the average forward delay varies wildly with a high frequency, the *Symmetry Ratio* obtained in this way can provide a reasonable performance.

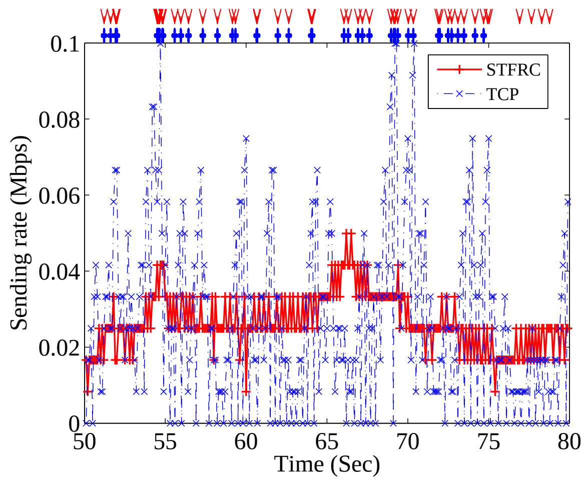


Figure 12. Sending rates and losses for the symmetric link case. Losses are shown at the top of the figure with v-shaped and + shaped symbols for STFRC and TCP, respectively.

Another important factor for STFRC for *Saratoga* is the interval of sending STATUS-feedback packets. For receiver-based TFRC, at least one feedback packet per RTT is recommended. For *Saratoga*, as we found, more than one STATUS-feedback packet per RTT may result in duplicate retransmissions [5] that are not reported if lost. This makes p to be estimated smaller than its actual value, making STFRC for *Saratoga* unfair to TCP. Therefore, we recommend one STATUS-feedback packet per RTT for STFRC for *Saratoga*. Sending only one STATUS-feedback packet per RTT benefits asymmetric and constrained return-channels. Sending a STATUS-feedback packet with an interval of more than an RTT will cause the STFRC to perform poorly as far as responding to the change of the network condition is concerned.

6. Conclusion

We have designed and presented a true Sender-based TCP-Friendly Rate Control (STFRC) for the *Saratoga* protocol. This STFRC uses the information contained in STATUS-feedback packets sent by the existing *Saratoga* receiver for the rate-control, and requires modifications within the *Saratoga* sender only. Modifications to the protocol are not required, and the asymmetric environments that *Saratoga* was designed for can still be supported. Performance evaluation with *ns*-2 simulations indicates that STFRC shares the bandwidth with TCP and co-existing flows fairly. Evaluation reveals the requirement for just enough bandwidth at the feedback path to allow a fair share to STFRC when the bandwidth is shared with TCP. These TFRC additions can enable *Saratoga* to be used safely across the public Internet.

Acknowledgments

This work is supported by NASA Grant NNX06AE44G and Cisco Systems Grant 2009-03096(3696).

References

[1] L. Wood, W. M. Eddy, C. Smith, W. Ivancic and C. Jackson, "Saratoga: A scalable file transfer protocol," work in progress as an Internet-draft, Dec. 2010.

[2] C. Underwood, S. Machin, P. Stephens, D. Hodgson, A. da Silva Curiel and M. Sweeting, "Evaluation of the utility of the Disaster Monitoring Constellation in support of Earth observation applications," in *IAA Symposium on Small Satellites for Earth Observation*, Berlin, Germany, Apr. 2005.

[3] W. Ivancic, W. M. Eddy, D. Stewart, L. Wood, J. Northam and C. Jackson, "Experience with delay-tolerant networking from orbit," *International Journal of Satellite Communications and Networking*, special issue for best papers of the *Fourth Advanced Satellite Mobile Systems Conference (ASMS 2008)*, vol. 28, no. 5-6, pp. 335–351, Sep./Dec. 2010.

[4] L. Wood, C. Smith, W. M. Eddy, W. D. Ivancic and C. Jackson, "Taking Saratoga from space-based ground sensors to ground-based space sensors," in *IEEE Aerospace Conference*, Big Sky, Montana, Mar. 2010.

[5] A. Z. M. Shahriar, M. Atiquzzaman and W. Ivancic, "Performance evaluation of NEMO in satellite networks," in *IEEE Military Communications (MILCOM)*, San Diego, California, Nov. 2008, pp. 1–7.

[6] K. Delin, Y. Chao and L. Lemmerman, "Earth science system of the future: observing, processing, and delivering data products directly to users," in *IEEE International Geoscience and Remote Sensing Symposium*, Sydney, Australia, Jul. 2001, pp. 429–431.

[7] L. Wood, W. Eddy, W. Ivancic, J. McKim and C. Jackson, "Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization," International Workshop on Space and Satellite Communications (IWSSC '07), Salzburg, Austria, Sep. 2007, pp. 163–167.

[8] J. Widmer, R. Denda and M. Mauve, "A survey on TCP-Friendly congestion control," *IEEE Network*, vol. 15, pp. 28–37, May/Jun. 2001.

[9] S. Floyd, M. Handley, J. Padhye and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348, Sep. 2008.

[10] S. Floyd, M. Handley, J. Padhye and J. Widmer, "Equation-based congestion control for unicast applications," in *ACM SIGCOMM*, Stockholm, Sweden, Aug./Sep. 2000, pp. 43–56.

[11] G. Renker and G. Fairhurst, "Sender RTT Estimate Option for DCCP," work in progress as an Internet-draft, Dec. 2010.

[12] G. Jourjon, E. Lochin and P. Sénac, "Towards sender-based TFRC," in *IEEE International Conference on Communications*, Glasgow, Scotland, Jun. 2007, pp. 1588–1593.

[13] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *ACM SIGCOMM*, Vancouver, Canada, Sep. 1998, pp. 303–314.

[14] K. Fall and K. Varadhan (eds.),"The Network Simulator *ns*-2: Documentation," http://www.isi.edu/nsnam/ns/ns-documentation.html.

Biography

Abu Zafar M. Shahriar received his BSc and MSc degrees from Bangladesh University of Engineering and Technology, Bangladesh in 1999 and 2004 respectively, both in Computer Science and Engineering. Currently he is a Research Assistant and working towards his PhD in the School of Computer Science at University of Oklahoma. His research interests include mobility of IPv6 networks in terrestrial and space networks, and file transfer protocols for satellite networks. He has several conference and journal papers published by IEEE and Springer.

Dr. Atiquzzaman is the Edith Kinney Gaylord Presidential Professor of School of Computer Science at The University of Oklahoma. He earned his M.Sc. and Ph.D. from the University of Manchester, England. Dr. Atiquzzaman is Editor-in-Chief of Journal of Network and Computer Applications and Co-Editor-in-Chief of Computer Communications Journal, and serves on the editorial boards of a number of journals including IEEE Communications Magazine He is the symposium co-chair of IEEE International Conference on Communications (2011) and IEEE Globecom Conference (2012) and SPIE Quality of Service over Next-Generation Data Networks Conference (2001,2002, 2003, 2005, 2006). He serves on the technical program committees of many national and international conferences, including IEEE INFOCOM, IEEE GLOBECOM, and IEEE International Conference on Computers and Communication Networks. His research is supported by over $3.2M grants from agencies such as National Science Foundation (NSF), National Aeronautics and Space Administration (NASA), U.S. Air Force, and Cisco. He has received the OU Regents' award for superior accomplishment in research and creative activity, NASA Group Achievement Award, IEEE Communication Society's Fred W. Ellersick Prize, and Institution of Electrical Engineers (IEE, UK) Premium Award.

Will Ivancic has over twenty-five years experience in network and system engineering for communication applications, communication networking research, state-of-the-art digital, analog and RF hardware design and testing. He is currently a senior research engineer at NASA's Glenn Research Center where he directs the hybrid satellite/terrestrial networking, space-based Internet, and aeronautical Internet research. He has led research efforts to deploy commercial-off-the-shelf (COTS) technology into NASA missions including the International Space Station and Shuttle. Mr. Ivancic has also performed joint research with Cisco Systems on advance routing techniques for space-based and aeronautic-based networks. Of particular interest is large scale, secure deployment of mobile networks, including Mobile IP and mobile router technology. Recent accomplishments include being first to demonstrate and deploy secure mobile networks in an operational government network, the US Coast Guard, first to deploy Mobile-IP Mobile networking on a space-based asset, the Cisco router in Low Earth Orbit (CLEO), first to deploy Internet Protocol security (IPsec) and Internet Protocol version 6 on a space-based asset, and first to deploy delay/disruption network technology bundling protocol in space.
Mr. Ivancic is also principal of Syzygy Engineering, a small consulting company specializing in communications systems and networking as well as advanced technology risk assessment. Mr. Ivancic is currently performing research and development on identity-based security and key and policy management and distribution for tactical networks – particularly mobile networks.

Lloyd Wood is a Chartered Engineer with experience in computing, networking and aerospace. As a space initiatives manager in Cisco Systems' Global Government Solutions Group, Lloyd had responsibility for CLEO, the Cisco router in Low Earth Orbit. Working with colleagues at NASA Glenn Research Center and Surrey Satellite Technology Ltd, Lloyd achieved the first tests from space of IPv6 and of the delay-tolerant networking bundle protocol for the "Interplanetary Internet." Lloyd gained his PhD from the Centre for Communication Systems Research at the University of Surrey, where he researched internetworking and satellite constellations, and to where he has returned as a research fellow.