An Implementation and Analysis of SCPS-TP in ns-3

Truc Anh N. Nguyen* and James P.G. Sterbenz*†§
*Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, Kansas 66045
†School of Computing and Communications (SCC) and InfoLab21
Lancaster University, LA1 4WA, UK
§The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
annguyen@ittc.ku.edu,jpgs@ittc.ku.edu

ABSTRACT

Given the importance of TCP in transport-layer protocol studies and the numerous TCP modifications, yet the limited TCP models in ns-3, we extend the existing TCP framework in the network simulator by implementing SCPS-TP, a transport-layer protocol for space communications. The TCP backward-compatible SCPS-TP is constructed as a set of TCP enhancements through the utilization of TCP options to address the unique characteristics of space networks with error-prone, highly asymmetric, and bandwidth-constrained channels. In this paper, we present our implementation together with a set of simulations to validate our model against the original SCPS-TP paper. Through the verification, we also analyze the performance of SCPS-TP in comparison with the standard TCP.

CCS CONCEPTS

•Networks → Transport protocols; Network simulations;

KEYWORDS

TCP, SCPS-TP, Space Communications, Vegas, SNACK, Transport Protocols, ns-3 Network Simulator, Performance Evaluation, SACK, NAK, Corruption, Congestion

ACM Reference format:
Truc Anh N. Nguyen and James P.G. Sterbenz. 2017. An Implementation and Analysis of SCPS-TP in ns-3. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 7 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067679

1 INTRODUCTION AND MOTIVATION

With its principal role in the Internet success and stability, the Transmission Control Protocol (TCP) [16] that was designed with a set of assumptions that are no longer valid in emerging networking, has undergone numerous modifications and extensions. Many of these changes improve TCP performance in environments that are characteristically different from the wired network for which TCP was originally tailored. The Space Communications Protocol Standards-Transport Protocol (SCPS-TP) [7] is such enhancement. SCPS-TP is the transport component of the SCPS protocol suite developed to address the challenges imposed by the space environment on reliable end-to-end (E2E) data communications. In addition to SCPS-TP, the suite also consists of file transfer, security, and network protocols. Designed to be backward-compatible with TCP, SCPS-TP utilizes TCP options to implement a set of mechanisms that augment TCP shortcomings in bandwidth-constrained and asymmetric space channels with high bit-error probability. Unlike TCP that always assumes congestion as the only cause of a packet loss, SCPS-TP realizes the different sources of data loss in space networks and designates distinct algorithms to cope with each loss type. In addition, the protocol is equipped with a header compression mechanism and hybrid negative acknowledgement (ACK) to cope with the limited bandwidth resource and uses a fixed ACK frequency to reduce the load on the return link, which generally has much lower capacity than the forward data link.

As we continue our effort to extend TCP models in the ns-3 network simulator [14], we implement SCPS-TP with the hope that our contribution will be helpful to the research community in studying transport-layer protocol behaviors using simulations. Furthermore, we anticipate the implementation of other satellite stack components following our SCPS-TP model that will enable more sophisticated and realistic space communication studies in ns-3. In this paper, we present our implementation together with a set of simulations to validate the SCPS-TP model against the original paper [7]. Through the evaluation, we analyze the performance of SCPS-TP in comparison with standard TCP.

The remainder of the paper is organized as follows: Section 2 discusses the drawbacks of TCP in space communication and the corresponding SCPS-TP enhancements followed by a brief survey of other previous SCPS-TP studies. Section 3 explains our implementation of the protocol and how the new model interacts with the existing TCP framework in ns-3. We verify the correctness of our model in Section 4 and conclude our paper in Section 5 with directions for future work.

2 BACKGROUND AND RELATED WORK

We begin this section by summarizing the TCP drawbacks in the space environment as highlighted in the SCPS-TP paper. We then

WNS3 2017, Porto, Portugal

DOI: http://dx.doi.org/10.1145/3067665.3067679

describe in more detail the set of TCP extensions incorporated into SCPS-TP to address these limitations. The section is concluded with a brief survey of some previous studies on SCPS-TP.

2.1 TCP Limitations in Space Communications

The space environment is characterized by error-prone, asymmetric, capacity-constrained channels, and intermittent connectivity. Due to the noisiness in space communication channels, bit errors are common, resulting in a high number of corruption-induced losses in addition to congestion-based. The TCP congestion control algorithm, without modifications and enhancements, fails to sustain the protocol throughput when operating over uncongested and noisy links due to its assumption that all packet losses are the consequence of network congestion. As a result of many engineering tradeoffs and the nature of scientific missions carried by space networks, space communication channels are highly asymmetric with substantially greater forward link capacity than the return link bandwidth. Because a TCP receiver usually acknowledges every other in-order segment, TCP overall throughput is constrained by the limited capacity of the return link. Moreover, because space networks comprise lower bandwidth links than wired networks, bit-efficiency becomes more important. TCP, with its 20-byte header per segment, creates significant overhead especially when small segments are used to reduce bit-error probability. The problem is intensified by the limited spacecraft power. Finally, connectivity within the space environment is usually intermittent with a dynamic network topology. TCP was designed for a connection with a stable E2E path and its ACK-clocking system will fail in the absence of a steady flow of acknowledgements. As a consequence, TCP suffers from poor throughput performance due to the high number of retransmissions. In the worst-case scenario when TCP exceeds its maximum retransmission threshold, the connection will be aborted.

2.2 SCPS-TP Enhancements on TCP

Corresponding to each characteristic of the space environment presented above is a set of enhancements employed by SCPS-TP.

SCPS-TP implements separate algorithms to deal with the three sources of packet loss existed in a space network: congestion, corruption, and link outage. The protocol employs two mechanisms for identifying the source of loss: default assumption and explicit signaling. The sender's default behavior is set by a network manager or an application and can be altered by an explicit notification from the data receiver or intermediate nodes including routers and ground stations. To cope with congestion-induced losses, SCPS-TP uses the delay-based TCP Vegas congestion control algorithm [1, 5]. The protocol employs the same Vegas slow-start algorithm to double the congestion window every other round trip time (RTT) while adding an additional congestion avoidance phase initiator. A SCPS-TP sender can also enter congestion avoidance when the congestion window reaches the network's bandwidth-×-delay product (BDP), which is a parameter with a supplied value. On the other hand, when a corruption-induced loss occurs, the sender neither invokes its congestion control algorithm nor backoffs its retransmission timer. The sender learns about corruption by the presence of a corruption-experienced option attached to an acknowledgement sent from the receiver, which in turn is notified by a corruption-experienced ICMP message from its ground station. To prevent overflowing the network link capacity when the congestion control algorithm is disabled, SCPS-TP uses a token-packet rate-control mechanism [15]. Finally, in response to a link outage, the sender enters persist mode during which it transmits periodic probe packets, suspends its retransmission timer, and ceases all data transmissions. The sender learns about link outage by receiving a link-outage ICMP message from its ground station.

To cope with asymmetric channels, SCPS-TP uses a fixed ACK frequency based on its estimated RTT, even when an out-of-order segment is received. The open-loop, token-packet rate-control mechanism is again used to clock out data transmission on the sender's side in the absence of a steady flow of acknowledgements. SCPS-TP also utilizes header compression to reduce the bandwidth usage on the ACK channel.

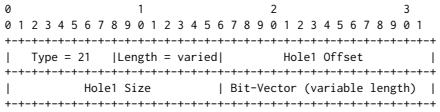

Figure 1: SNACK option format [4]

To cope with bandwidth-constrained channels, SCPS-TP implements two mechanisms: header compression and the selective negative acknowledgement (SNACK) option. The SCPS-TP E2E header compression technique replaces the port numbers in the regular header with a connection identifier and omits any fields that are irrelevant to the segment being transmitted. This header compression mechanism is loss-tolerant since the decompression of a packet is independent from the decompressions of the preceding ones. SCPS-TP communicating entities negotiate their use of header compression by appending an option to the initial SYN and SYN-ACK signaling segments. SCPS-TP selective negative acknowledgement SNACK is a hybrid version of the selective acknowledgement SACK [12] and negative acknowledgement NAK [9] designed to recover from multiple losses per sending window in a bit-efficient way. Since the receipt of a SNACK option is equivalent to an immediate retransmission request, SCPS-TP eliminates the reliance on the Fast Retransmit algorithm, which may never be triggered when the ACK frequency is tuned to accommodate the limited return link capacity. Figure 1 illustrates the format of a SNACK option as defined in the CCSDS Recommended Standard for SCPS-TP [4]. The Type and Length fields are mandatory in all TCP options. The SNACK option has a type of 21 with variable length if the bit-vector field is presented; otherwise the option length is 6 octets. The Hole1Offset field holds the offset from the current ACK number of the first hole being reported in the option, while the Hole1Size field holds the size of the first hole in maximum-sized segment (MSS) units. Finally, the Bit-Vector field is used to report additional missing data in the receiver's buffer following the first hole.

Furthermore, SCPS-TP employs the Timestamps option [11] for a more accurate RTT estimation and the Window Scaling option [11] to allow new data transmissions on the sender side while it is trying to recover from corruption-induced losses.

2.3 Related Work

The performance of SCPS-TP has been studied extensively since its inception. Analysis of the protocol using various network scenarios includes under the impact of long propagation delay [18], over a long-delay, error-prone cislunar communication links [17], and over NASA geostationary orbit (GEO) Advanced Communications Technology Satellite (ACTS) with the particular focus on the effectiveness of those SCPS-TP mechanisms designed to cope with space channel asymmetry [19].

3 SCPS-TP IMPLEMENTATION IN NS-3

Our implementation of SCPS-TP in ns-3 follows the CCSDS recommended standard for SCPS transport protocol [4]. As a protocol that is comprised of multiple extensions to TCP, the SCPS-TP model also resides in the Internet module along with TCP models.

3.1 Data Loss Handling

The SCPS-TP approach for dealing with different sources of data loss in space communications is implemented inside the `ns3::ScpsTpSocketBase` class, which is an inheritance of `ns3::TcpSocketBase`. The sender's default behavior is set to corruption and can be changed to either congestion or link outage by tuning the **m_lossType** attribute. Our SCPS-TP model allows the use of any TCP congestion control algorithm implemented in ns-3 when **m_lossType** is set to congestion. As the first milestone of our SCPS-TP implementation project, we implemented TCP Vegas [13], which is now part of the standard ns-3 release.

When the **m_lossType** attribute is set to corruption, upon a requirement for data retransmission, i.e. when the number of received duplicate ACKs has reached its threshold defined in **m_retxThresh**, when the sender receives a SNACK, or upon the expiration of the retransmission timer, SCPS-TP does not modify its congestion window **m_cWnd** and slow-start threshold **m_ssThresh** values. This mechanism requires SCPS-TP to overwrite the `TcpSocketBase::ReTxTimeout()` and `TcpSocketBase::EnterRecovery()` functions.

3.2 Fixed ACK Frequency

Unlike TCP that transmits an ACK for every other segment received, a SCPS-TP receiver acknowledges data on a fixed frequency. This mechanism is handled by the `ScpsSocketBase::ReceivedData()` function. The receiver is not allowed to ACK any received data, including out-of-order sequences, before the delay ACK timer is fired. The duration of this timer is the value of the tunable parameter **m_delAckTimeout** used in the standard TCP.

3.3 SNACK Implementation

Similar to other existing TCP options in ns-3, the SNACK option is defined inside the `ns3::ScpsTpOptionSnack` as a child class of the `ns3::TcpOption`. In addition to the `ScpsTpOptionSnack::Serialize()` and `ScpsTpOptionSnack::Deserialize()` function that writes the option to and extracts (and reconstructs) it from the byte buffer of the packet to which the option is appended, respectively, the class also defines a SNACK hole as a pair of sequence numbers. A hole is a block of one or more contiguous missing segments in the receiver's buffer. The first sequence number is used to represent the left edge while the second sequence number is used to represent the right edge of the hole. Here, we borrow the representation of a SACK block from RFC 2018 [12] in our implementation of a SNACK hole except that the sequence number pair in SNACK represents missing instead of queued data because SNACK is a negative acknowledgement. All existing holes in the receiver's buffer at a given time is stored in a SNACK list. This list allows the transmissions of multiple SNACK options in a single ACK packet. The goal is to ensure that our SNACK implementation is compatible with the SACK code that has been recently pushed into the ns-3 development tree and will be part of the next ns-3.27 standard release.

```
TcpOptionSack::SackList::iterator it=m_sackList.begin();
TcpOptionSack::SackBlock begin = *it;
ScpsTpOptionSnack::SnackHole hole;
++it;
// We iterate through the currently updated sack list to
// construct snack list
while (it != m_sackList.end()){
TcpOptionSack::SackBlock next = *it;
hole = ScpsTpOptionSnack::SnackHole (next.second, begin.
    first);
m_snackList.push_front (hole);
begin = *it;
++it;
}
/* Now we need to push one more hole to the front of
* snack list. This is the hole that has offset 0 from
* the current ACK number */
hole = ScpsTpOptionSnack::SnackHole (m_nextRxSeq.Get (),
    begin.first);
m_snackList.push_front (hole);
```

Listing 1: TcpRxBuffer::UpdateSnackList()

Similar to SACK, the SNACK list generation is performed inside `ns3::TcpRxBuffer`. Since a SNACK list is viewed as a complement of a SACK list, the former can be constructed directly from the latter. After the SACK list is generated or updated through the `TcpRxBuffer::UpdateSackList()` private function when an out-of-order segment arrives at the receiving buffer, the SNACK list is also generated or updated through the function `TcpRxBuffer::UpdateSnackList()` as shown in Listing 1. Upon the arrival of a new segment that advances **m_nextRxSeq**, holes that have already been fixed should be removed from the SNACK list through the `TcpRxBuffer::ClearSnackList()` private function as presented in Listing 2, similar to the way old data blocks are removed from the SACK list through `TcpRxBuffer::ClearSackList()`. Objects from other classes can obtain the current SNACK list using the public function `TcpRxBuffer::GetSnackList()`. When the receiver transmits an ACK using `ScpsTpSocketBase::SendEmptyPacket()`, if the SNACK list is not empty, the receiver will call the `ScpsTpSocketBase::AddOptionSnack()` function repeatedly to construct multiple SNACK options until either there are no items left in the list or the maximum allowed number of TCP option octets has been reached. A code snippet of the `SendEmptyPacket()` function that demonstrates how SNACK options are appended to an ACK header is shown in Listing 3.

```
ScpsTpOptionSnack::SnackHole hole = *it;
if (hole.second <= seq){
  it = m_snackList.erase (it);
}
```

```
else if ((hole.first < seq) && (seq < hole.second)){
  hole.first = seq;
}
```

Listing 2: TcpRxBuffer::ClearSnackList()

```
if (m_snackEnabled && m_rxBuffer->GetSnackListSize () >
    0)
{
  ScpsTpOptionSnack::SnackList snackList = m_rxBuffer->
    GetSnackList ();

  // Calculate the number of SNACK options can be
    appended to this ACK
  uint8_t optionLenAvail = header.GetMaxOptionLength () -
     header.GetOptionLength ();
  uint8_t allowedSnackOptions = optionLenAvail / 6;

  ScpsTpOptionSnack::SnackList::iterator i;
  uint16_t hole1Offset, hole1Size;

  /* Now we iterate through the SNACK list,
   * calculate the offset and size of each hole,
   * and construct a SNACK option for that hole */
  for (i = snackList.begin (); allowedSnackOptions > 0 &&
      i != snackList.end (); ++i)
  {
    hole1Offset = ((*i).first - m_rxBuffer->
     NextRxSequence ()) / m_tcb->m_segmentSize;
    hole1Size = ((*i).second - (*i).first) / m_tcb->
     m_segmentSize;
    AddOptionSnack (header, hole1Offset, hole1Size);
    allowedSnackOptions--;
  }
}
}
```

Listing 3: ScpsTpSocketBase::SendEmptyPacket()

SNACK options, when arriving at the data transmitter, are processed by the `ScpsTpSocketBase::ProcessOptionSnack()` function where all the reported holes (one hole per option when **Bit-Vector** is not used and an ACK can carry multiple SNACK options) are placed in a received SNACK list. This list is then passed as an argument to the `UpdateSnackedData()` function inside the `ns3::TcpTxBuffer` class. The function is responsible for iterating through the transmit buffer and marking those sequence numbers reported as missing in the recently received SNACK for immediate retransmissions.

4 VERIFICATION AND ANALYSIS

The lack of an available implementation of SCPS-TP in the Linux kernel prevents us from using DCE [6] to verify the correctness of our model. Thus, we have decided to validate our code by following the testing scenarios presented in the original paper [7]. Specifically, we reproduce the asymmetry and corruption tests conducted in the laboratory with the understanding that some differences in the results obtained from our simulations and their emulations are inevitable. In addition, we also evaluate the performance of SCPS-TP under the impact of long propagation delay and small segment size.

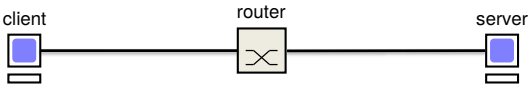

Figure 2: Topology for SCPS-TP vs. TCP comparison

4.1 Simulation Topology

The topology that we use is illustrated in Figure 2, in which a data-transmitting client communicates with a data-receiving server through a router. This router serves the same purpose as the Spanner program that the SCPS-TP emulation used to simulate a satellite channel. The link between the router and the server is configured to characterize a satellite link with different data rates, propagation delays, and bit error rates based on a specific testing scenario. The router implements a drop-tail queue with capacity of one bandwidth-×-delay product (BDP). We use `BulkSendApplication` to transmit 5 MB of data from the client to the server in each simulation. While SCPS-TP experiment used the SunOS 4.1.3 Unix kernel's TCP implementation in their comparison with SCPS-TP, we use TCP NewReno since it is the standard baseline TCP implementation available in the current ns-3 release.

4.2 Asymmetry Test

Table 1: SCPS-TP configurations for asymmetry tests

Parameter	Setting
Send buffer size	300 kB
Receive buffer size	300 kB
Congestion control	on
ACK frequency	0.1 s
Rate control	1.5 Mb/s
SNACK option	on, no bit-vector
Window Scaling option	on
TCP Timestamps option	off
SCPS-TP Header Compression	off

The asymmetry test demonstrates the performance of SCPS-TP in comparison with TCP when operating over asymmetric space communication channels. The channel connecting the sender and the router has a bandwidth of 1.5 Mb/s and a negligible delay of 0.01 ms. Each packet transmitted from the sender to the receiver has an MTU size of 512 bytes. The simulated space link between the router and the server is configured with a 50-ms propagation delay and a 1.5-Mb/s data rate in the forward direction. The return direction takes on each bandwidth value in the set [1.5 Mb/s, 15 kb/s, 5 kb/s, 3 kb/s, 2 kb/s, 1.5 kb/s]. This is equivalent to data channel and ACK channel bandwidth ratios ranging from 1:1 to 1000:1. The SCPS-TP configuration is summarized in Table 1, which is almost identical to Table 1 in the original paper, except the ACK frequency. In these simulations, SCPS-TP utilizes the Vegas congestion control algorithm. We disable the Timestamps and SACK options when simulating the standard TCP.

Figure 3 plots the average throughput of SCPS-TP in comparison with TCP NewReno over a range of ACK channel capacities, while Figure 4 shows transmitted sequence numbers as a function

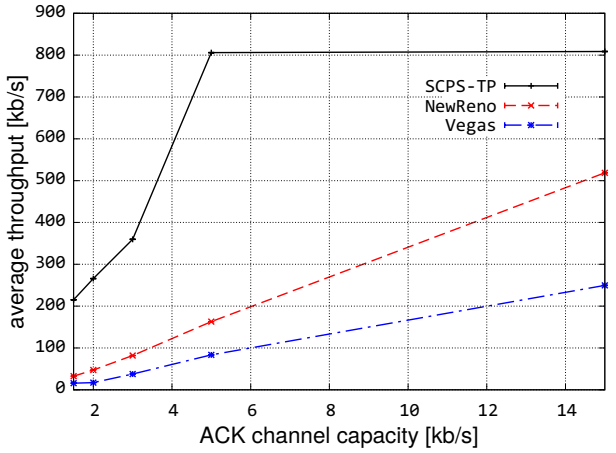

Figure 3: Asymmetric channel: average throughput

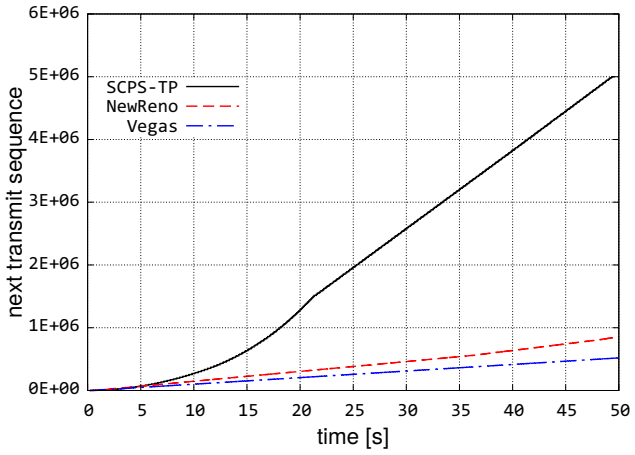

Figure 4: Asymmetric channel: tx sequence vs. time

of simulation time at the 5 kb/s ACK-link capacity for the protocols. We also simulate Vegas in this scenario to clearly demonstrate the effectiveness of the fixed and reduced ACK frequency approach employed by SCPS-TP in coping with asymmetric channels. The configuration of SCPS-TP shown in Table 1 is basically a combination of Vegas, fixed ACK, and SNACK.

Overall, we see that the higher the data-channel to ACK-channel bandwidth ratio, the lower the throughput achieved by the protocols. The throughput performance of the two TCP variants and the TCP-inherited SCPS-TP are all constrained by the limited reverse link capacity. TCP relies on ACK reception to utilize the network bandwidth. The low ACK-channel capacity limits the congestion window growth, resulting in a throughput degradation. Moreover, when the forward to return link bandwidth ratio is high, the reverse channel is saturated before the forward link causing some ACKs to be dropped at the queue. However, the throughputs of NewReno and Vegas drop much faster than the SCPS-TP throughput. At the 300:1 ratio (at the 5 kb/s ACK channel bandwidth), the throughput achieved by SCPS-TP is approximately 800 kb/s, which is about 5 times higher than the throughput achieved by NewReno and almost 10 times higher than the throughput achieved by Vegas. The better performance of SCPS-TP owes to its reduced ACK frequency mechanism. As highlighted in the SCPS-TP paper, the ACK frequency needs to be properly tuned and merits further study. When we use the ACK frequency to be approximately 50% of the available ACK channel capacity as suggested for this scenario, we see significant fluctuations in the SCPS-TP throughput. Many simulations with different ACK frequency values lead us to our use of 0.1 s delayed ACK in these asymmetry tests.

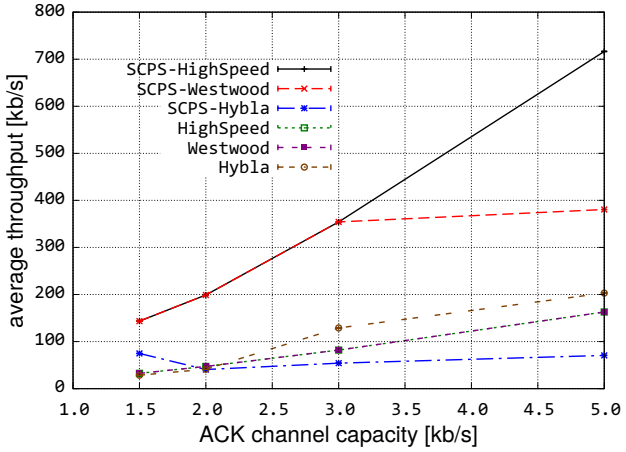

Figure 5: Asymmetric channel: average throughput

We extend our study of the reduced ACK frequency benefit in coping with bandwidth asymmetry by simulating SCPS-TP with other congestion control algorithms and comparing these SCPS-TP variants with the corresponding TCP variants. We choose Westwood [3, 10], HighSpeed [8], and Hybla [2] for our evaluation. As shown in Figure 5, while the performance of both Westwood and HighSpeed are improved with the employment of the fixed ACK approach, the performance of Hybla is not.

4.3 Impact of Delay Test

In the second set of simulations, we study the impact of long propagation delay on the performance of SCPS-TP and NewReno. The simulated space link between the router and the receiver has a delay varied from 50 ms to 300 ms. The SCPS-TP and TCP configurations as well as the other simulation parameters are the same as in the previous asymmetry test.

Figure 6 plots the average throughput of SCPS-TP and NewReno at the 1:1 and 500:1 forward to return link bandwidth ratios as a function of the space channel delay. Overall, as the delay increases, the throughput performance decreases. The longer it takes for the data and their ACKs to traverse through the links, the slower the congestion window grows. When the ACK and data channels have the same capacity, i.e. when the ratio is 1:1, NewReno achieves better performance than SCPS-TP. At the 100 ms delay, the throughput achieved by NewReno is about 1.1 Mb/s, twice the throughput

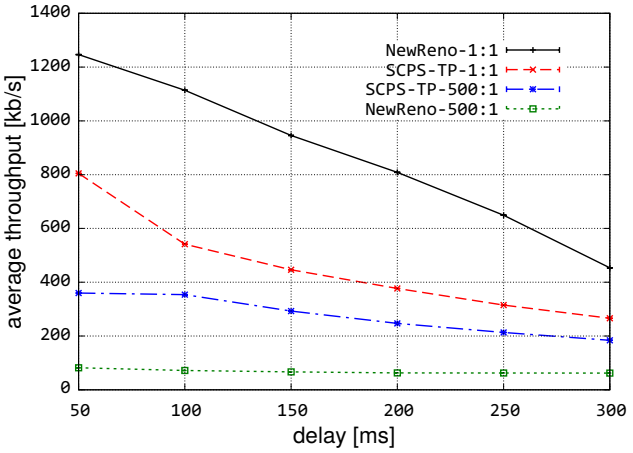

Figure 6: Impact of long delay: Average throughput

achieved by SCPS-TP. However, when the ratio increases to 500:1, SCPS-TP outperforms NewReno. At the 100 ms delay, SCPS-TP obtains a throughput of 354 kb/s while NewReno can only acquire a throughput of 82 kb/s.

4.4 Impact of Segment Size Test

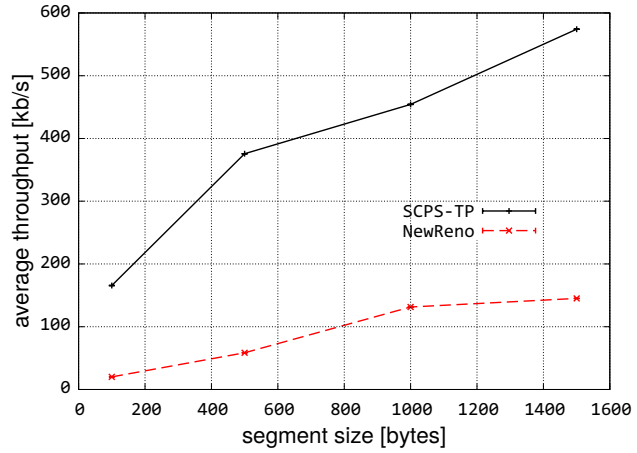

Figure 7: Impact of segment size: Average throughput

In the third set of simulations, we study the impact of segment size on the performance of SCPS-TP and NewReno. Again, we use the same setting as in the previous tests, except that we fix the reverse link capacity at 3 kb/s (equivalent to 500:1 ratio) while varying the MTU size from 100 bytes to 1500 bytes.

Figure 7 plots the average throughput of the two protocols as a function of MTU size. We see that segment size has a strong effect on the performance since both protocols achieve much better throughput with large MTU size. Small-sized packets prevent the protocols from fully utilizing the network available bandwidth. SCPS-TP again outperforms NewReno.

4.5 Corruption Test

The corruption test demonstrates the performance of SCPS-TP in comparison with TCP when operating over lossy space communication channels. The simulated space link between the router and the server is configured with a 10^{-5} bit-error rate (BER) using `ns3::RateErrorModel`. This link has a symmetric bandwidth of 1.5 Mb/s in both directions. Two independent simulations are performed; one with a 50 ms and the other with a 100 ms propagation delay for the satellite channel. The SCPS-TP configuration for these corruption tests is summarized in Table 2, which is reproduced from Table 2 in the original paper.

:

Table 2: SCPS-TP configurations for corruption tests

Parameter	Setting
Send buffer size	300 kB
Receive buffer size	300 kB
Congestion control	off
ACK frequency	1 ACK every 69 ms
Rate control	1.425 Mb/s
SNACK option	on, no bit-vector
Window Scaling option	on
TCP Timestamps option	off
SCPS-TP Header Compression	off

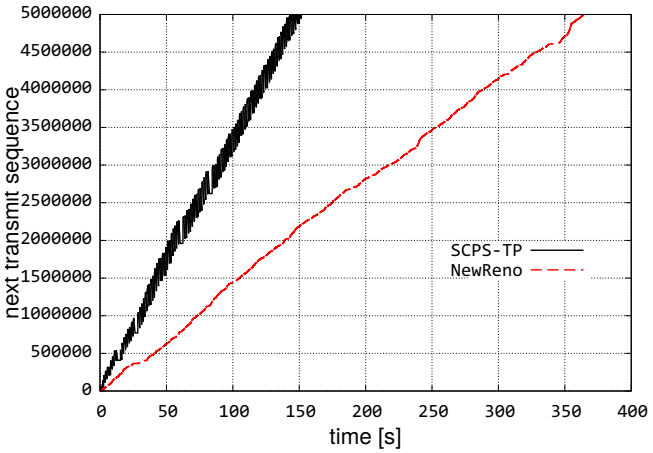

Figure 8: Corruption: Tx sequences vs. time at 50 ms delay

Figure 8 and Figure 9 show transmitted sequence numbers by both protocols as a function of simulation time at the 50 ms and 100 ms delay, respectively. We can see that SCPS-TP is able to transmit more data faster than NewReno because when corruption is enabled, while NewReno blindly halves its congestion window, SCPS-TP does not reduce its sending rate upon a packet loss event. Moreover, while NewReno throughput drops from 110 kb/s to 58 kb/s when the propagation delay increases from 50 ms to 100 ms, SCPS-TP throughput stays the same at about 300 kb/s.

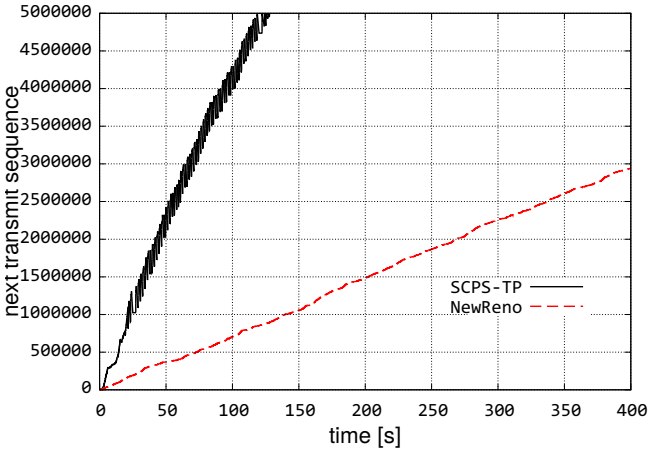

Figure 9: Corruption: Tx sequences vs. time at 100 **ms delay**

5 CONCLUSIONS

In this paper, we present our ns-3 model of SCPS-TP, which is comprised of a set of TCP extensions to address the unique characteristics of the space environment. While verifying the correctness of our implementation by reproducing the experiments presented in the original SCPS-TP paper by Durst et al., we study the performance of the protocol under various network conditions. In the asymmetry test, we prove the effectiveness of the fixed and reduced ACK frequency mechanism employed by SCPS-TP in coping with high forward and return channel bandwidth ratios, which is also highlighted in the original SCPS-TP paper. In addition, we see that in order to achieve the benefit from this mechanism, the ACK frequency needs to be properly tuned for each scenario. The potential of the SCPS-TP's ACK technique is also valid when SCPS-TP employs some congestion control algorithms different from Vegas. Furthermore, when the ACK frequency is reduced, SCPS-TP is able to achieve better throughput than NewReno when the asymmetry couples with long delay and/or small segment MTU size. Last but not least, we show the effectiveness of SCPS-TP corruption handling mechanism in the corruption test.

For future work, we plan to implement the link outage algorithm since it is the only missing component in the current SCPS-TP model. In addition, we plan to extend our study of the protocol under additional network scenarios. We are also interested in comparing the performance of SNACK and SACK under error-prone channels.

ACKNOWLEDGMENTS

The authors would like to acknowledge the members of the ResiliNets research group for their useful discussions and suggestions that helped us with this implementation. The authors would like to thank the anonymous reviewers for their helpful feedback on this paper. Finally, the authors would also like to thank the ns-3 development team for their timely responsiveness to guidance and issues with the ns-3 platform. This work was funded in part by NSF grant CNS-1219028 (Resilient Network Design for Massive Failures and Attacks).

REFERENCES

[1] L. Brakmo, S. O'Malley, and L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *SIGCOMM Comput. Commun. Rev.* 24, 4 (1994), 24–35.

[2] C. Caini and R. Firrincieli. 2004. TCP Hybla: a TCP Enhancement for Heterogeneous Networks. *International Journal of Satellite Communications and Networking* 22, 5 (2004), 547–566. DOI: http://dx.doi.org/10.1002/sat.799

[3] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang. 2002. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks* 8, 5 (2002), 467–479.

[4] CCSDS-The Consultative Committee for Space Data Systems. 2006. Space Communications Protocol Specification (SCPS)-Transport Protocol (SCPS-TP). http://public.ccsds.org/publications/archive/714x0b2.pdf. (October 2006). Issue Recommended Standard, Issue 2. http://public.ccsds.org/publications/archive/714x0b2.pdf

[5] P. Danzig, Z. Liu, and L. Yan. 1995. An Evaluation of TCP Vegas by Live Emulation. (1995).

[6] DCE. 2011. Direct Code Execution. https://www.nsnam.org/overview/projects/direct-code-execution/. (July 2011).

[7] R. Durst, G. Miller, and E. Travis. 1996. TCP Extensions for Space Communications. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*. ACM Press, New York, NY, USA, 15–26.

[8] S. Floyd. 2003. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental). (Dec. 2003). http://www.ietf.org/rfc/rfc3649.txt

[9] R. Fox. 1989. TCP Big Window and NAK Options. RFC 1106. (June 1989). http://www.ietf.org/rfc/rfc1106.txt

[10] S. Gangadhar, T. A. Nguyen, G. Umapathi, and J. Sterbenz. 2013. TCP Westwood Protocol Implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*. Cannes, France.

[11] V. Jacobson, R. Braden, and D. Borman. 1992. TCP Extensions for High Performance. RFC 1323 (Proposed Standard). (May 1992). http://www.ietf.org/rfc/rfc1323.txt

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. 1996. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard). (Oct. 1996). http://www.ietf.org/rfc/rfc2018.txt

[13] T. A. Nguyen, S. Gangadhar, M. Rahman, and J. Sterbenz. 2016. An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3. In *Proceedings of the 2016 Workshop on ns-3 (WNS3 '16)*. ACM, Seattle, United States, 8.

[14] ns-3. 2009. The ns-3 Network Simulator. http://www.nsnam.org. (July 2009).

[15] C. Partridge. 1994. *Gigabit Networking*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[16] J. Postel. 1981. Transmission Control Protocol. RFC 793 (Standard). (Sept. 1981). http://www.ietf.org/rfc/rfc793.txt Updated by RFCs 1122, 3168.

[17] R. Wang, N. Aryasomayajula, A. Ayyagari, and Q. Zhang. 2007. An Experimental Performance Evaluation of SCPS-TP over Cislunar Communications Links. In *Wireless Communications and Networking Conference, 2007. WCNC 2007*. IEEE, 2603–2607.

[18] R. Wang, V. Bandekodige, and M. Banerjee. 2004. An Experimental Evaluation of Link Delay Impact on Throughput Performance of TCP and SCPS-TP in Space Communications. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, Vol. 6. 4061–4065. DOI: http://dx.doi.org/10.1109/VETECF.2004.1404841

[19] R. Wang and S. Horan. 2009. Protocol Testing of SCPS-TP over NASA's ACTS Asymmetric Links. *IEEE Trans. Aerospace Electron. Systems* 45, 2 (April 2009), 790–798. DOI: http://dx.doi.org/10.1109/TAES.2009.5089562