Network and Communication Technologies; Vol. 1, No. 2; 2012
ISSN 1927-064X E-ISSN 1927-0658
Published by Canadian Center of Science and Education

Timestamp Retransmission Algorithm for TCP-Cherry over InterPlaNetary Internet

Satoshi Utsumi[1] & Salahuddin Muhammad Salim Zabir[2]

[1] Department of Control and Information System Engineering, Tsuruoka National College of Technology, Yamagata, Japan

[2] Orange Labs, France Telecom, Tokyo, Japan

Correspondence: Satoshi Utsumi, Department of Control and Information System Engineering, Tsuruoka National College of Technology, Yamagata, Japan. Tel: 81-235-25-9077. E-mail: u-satoshi@tsuruoka-nct.ac.jp

Received: September 5, 2012 Accepted: October 26, 2012 Online Published: November 12, 2012

doi:10.5539/nct.v1n2p59 URL: http://dx.doi.org/10.5539/nct.v1n2p59

Abstract

InterPlaNetary (IPN) Internet links have extremely long propagation delays and very high link error rates. In such networks, lost retransmissions of the TCP data segments are common. Existing TCP schemes can detect lost retransmissions only through TCP retransmission timeout and hence are inefficient in such scenarios. In this paper we propse a new algorithm for handling such lost retransmission efficiently for TCP-Cherry over IPN Internet. Simulations show that our algorithm improves TCP-Cherry goodput by upto 13 times under high link error.

Keywords: InterPlaNetary Internet, congestion control, TCP-Cherry, TCP timestamp option, TCP retransmission

1. Introduction

With the increase in space missions like human transport to the International Space Station, Mars explorers and commercial space flights, the concept of InterPlaNetary (IPN) Internet has been becoming popular in recent years. The IPN Internet refers to connecting more than one network each of which is separated from others by stellar distance. Such a challenging networking environment is characterized by extremely long propagation delay, low bandwidth and very high link error rates (Akan et al., 2004).

In such scenarios, long propagation delay and high link errors often lead to repeated congestion detection by connection oriented protocols like TCP. Also, often the retransmitted segments get lost in the network. Several mechanisms have been proposed to address issues related to the end to end connectivity over the IPN Internet. An essential requirement for such schemes should be the ability to deploy over both the intra-planetary and the inter-planetary networking scenarios.

With a view to addressing the above requirements, in this paper, we propose a new algorithm to deploy along with TCP-Cherry (Utsumi et al., 2008; 2009). TCP Cherry is already known for its robustness compared with other mechanisms over satellite networks with long propagation delays and high link errors. The new algorithm ensures efficient handling of lost retransmissions under the harsh IPN conditions. As such, the performance of TCP Cherry is optimized for intra-Planetary and inter-Planetary scenarios. Simulation results show that our algorithm can improve the goodput yield of TCP-Cherry up to 13 times in high link error cases. Our algorithm can also be deployed with TP-Planet (Akan et al., 2004), but performance gains are lower due to *NIL* segment overheads.

The rest of the manuscript is organized as follows. In section 2, we discuss existing mechanisms that address end to end connectivity for IPN Internet. We propose our timestamp retransmission algorithm in section 3. In section 4, we evaluate our proposed algorithm by simulation. Finally, we conclude the manuscript in section 5.

2. Related Works

SCPS-TP (Wang et al., 2007) is a transport protocol for space communications. It has the selective negative acknowledgement (SNACK) option to detect lost segments. However, the SNACK option for SCPS-TP cannot detect lost retransmissions.

LTP (Wang et al., 2011) is also for space communications. It detects lost segments with the “report segments” for reliable parts of blocks (red parts). Report segments acknowledge selectively for data segments. Like acknowledgement mechanisms for traditional TCPs, report segments for LTP cannot detect lost retransmissions.

As the algorithms to detect lost retransmissions, duplicate acknowledgement counting (DAC) (Kim & Lee, 2004) and SACK+ (Kim et al., 2004) have been proposed. The DAC algorithm detects lost retransmissions by counting the number of duplicate acknowledgements. The SACK+ detects lost retransmissions by investigating SACK blocks. As well as TCP-Peach (or TCP-Peach+), TCP-Cherry uses low-priority segments with data block a lot. Then, DAC or SACK+ cannot apply to TCP-Cherry for detecting lost retransmissions deterministically, because of indeterminate arriving acknowledgements (duplicate acknowledgements or SACKs) for the low-priority segments. That is, the ACKs for the low-priority segments may lead to miss-counting the number of duplicate acknowledgements at the DAC algorithm. The ACKs for the *supplement* segments of TCP-Cherry may acknowledge unexpected data blocks for the SACK+ algorithm. For example, the ACKs for the supplement segments transmitted before detecting the first lost of the data segment may acknowledge the data blocks which sequence numbers are greater than the sequence number detecting the lost retransmission for SACK+.

3. Our Proposal

In the condition where the probability of segment loss due to link errors is large, duplicate segment losses, which are losses of the segments retransmitted by TCP due to three duplicate acknowledgements (dupacks), occur frequently. Conventional TCP can detect such duplicate segment losses only by TCP RTO (Retransmission Timeout). The high error rate over IPN increases the probability of lost retransmissions of the TCP data segments dramatically. Without special arrangements, a lost retransmission leads to TCP Retransmission Timeout (RTO) which adversely affects TCP performance.

For IPN, RTO can be significantly high. Hence, waiting for RTO to occur leads to delay in recovering from the loss. Despite the *Fast-Forward Start* algorithm in TCP-Cherry, this may lead to considerable inefficiencies. In addition, for multiple lost retransmissions in a single window, the effect gets even worse. To overcome this problem and thus recover multiple lost retransmissions in one window size during one RTT, we propose a new retransmission algorithm for handling such losses. We name this as the *Timestamp Retransmission* algorithm (Figure 1).

Our algorithm slightly modifies the mechanisms of original timestamp option in RFC 1323 (Jacobson et al, 1992) without changing its functionalities. In each dupack, the receiver echoes the Timestamp value (TSval) corresponding to the segment for which the dupack (SACK) is generated. However, the sender ignores the Timestamp echo reply (TSecr) field for RTT calculation purposes unless the acknowledgement advances the left edge of the window (i.e., new segment is being acknowledged). This tricky modification ensures full functional conformance to the provisions of RFC 1323.

To accomplish the above, the sender stores the sequence numbers and corresponding TSvals of the retransmitted segments in a table called *Timestamp Retransmission List*.

```
Timestamp_Retransmission(TSval, TSecr)
 if TimstampRetransmissionList is not Null
  for i = CummulativeAckedNo to LargestSackedNo
   if (There is TimestampRetransmissionList(i)) &
        (TimestampRetransmissionList(i) < TSecr)
    if Seqno i is Acked/Sacked
      RemoveTimestampRetransmissionList(i);
    else
      UpdateTimestampRetransmissionList(i, TSval);
      Retransmission(i);
    end;
   end;
  end;
 end;
end;
```

Figure 1. Timestamp retransmission algorithm

3.1 Timestamp Retransmission Algorithm

We explain the algorithm of Figure 1 here. The parameters and functions used in the algorithm are described as follows.

- a parameter *TSval* is the timestamp value for the next transmitted segment.
- a parameter *TSecr* is the timestamp echo of the received segment at that time.
- a function *TimestampRetransmissionList(x)* returns the TSval corresponding to the retransmitted data segment of the sequence number *x* in the *TimestampRetransmissionList*.
- *RemoveTimestampRetransmissionList(x)*, a function that removes the combinations of the sequence number *x* and the corresponding TSvals.
- *UpdateTimestampRetransmissionList(x, y)*, a function that updates the TSval for the sequence number *x* to *y*.
- *Retransmission(x)*, a function that retransmits the data segment corresponding to the sequence number *x*. Upon receiving a SACK, TCP sender compares the TSvals of the *(sequence numbers, TSvals of retransmitted data segments)* tuples in the *Timestamp Retransmission List* with the TSecr in the SACK.
- If all of the TSvals in *Timestamp Retransmission List* are larger than the TSecr in the SACK, the TCP sender does nothing about *Timestamp Retransmission List* (Case 1).
- If some TSvals in *Timestamp Retransmission List* are smaller than TSecr in the SACK and the SACK acknowledges the sequence numbers corresponding to the TSvals in the list, the TCP sender removes the corresponding sequence numbers-TSvals entries (Case 2).
- If some TSvals in *Timestamp Retransmission List* are smaller than TSecr in the SACK and the SACK does not acknowledge the sequence numbers corresponding to the TSvals in the list, the TCP sender retransmits the data segment corresponding to the sequence number without waiting for timeout and updates the TSvals for the sequence numbers in the table accordingly (Case 3).

During the first retransmission of the data segments, available bandwidth is estimated by corresponding low-priority *supplement segments* and corresponding congestion control actions are taken by the TCP sender. Hence, even in Case 3, the sender does not shrink the congestion window or transmit *supplement segments*. It infers that the loss of the retransmitted segment is due to link errors rather than congestion.

3.2 Timestamp Retransmission Example Scenario

Figure 2 shows an example of how a timestamp retransmission occurs. In this figure, rather than representing with real sequence number, we show a block of data using a single sequence number for easy understanding. Let us assume that at time t_0 (Figure 2) the sender transmits the data segment which will be lost in the network. We assume *maxcwnd*=64.

- At time $t=t_0$:

The sender transmits the data segment which will be lost in the network.

- At time $t=t_1$ (where, $t_1 \fallingdotseq t_0+RTT$):

According to 3 dupacks (SACKs), the sender retransmits the data segments which was lost, and will be lost again in the network. The sender restores the combination of the sequence number (1028) and TSval (10068) in the retransmitted segment in *Timestamp Retransmission List*.

- At time $t=t_2$(where, $t_2 \fallingdotseq t_0+2\times RTT$):

Since the TSval (10068) for the retransmitted segment with the sequence number 1028 in the *Timestamp Retransmission List* is smaller than the TSecr (10069) of the received SACK corresponding to the *supplement segment* with the sequence number 1130, the sender retransmits again the data segment with the sequence number 1028 without waiting for timeout.

In our proposal, TCP-Cherry sender needs the storage memory for Timestamp Retransmission List. Timestamp Retransmission List stores the information for the retransmitted segment in current congestion window. Then, Timestamp Retransmission List needs only the storage size $S < O(W) = O(RTT)$. Here, W is the congestion window size, which is linear for RTT. That is, the needed storage size is scalable for RTT.

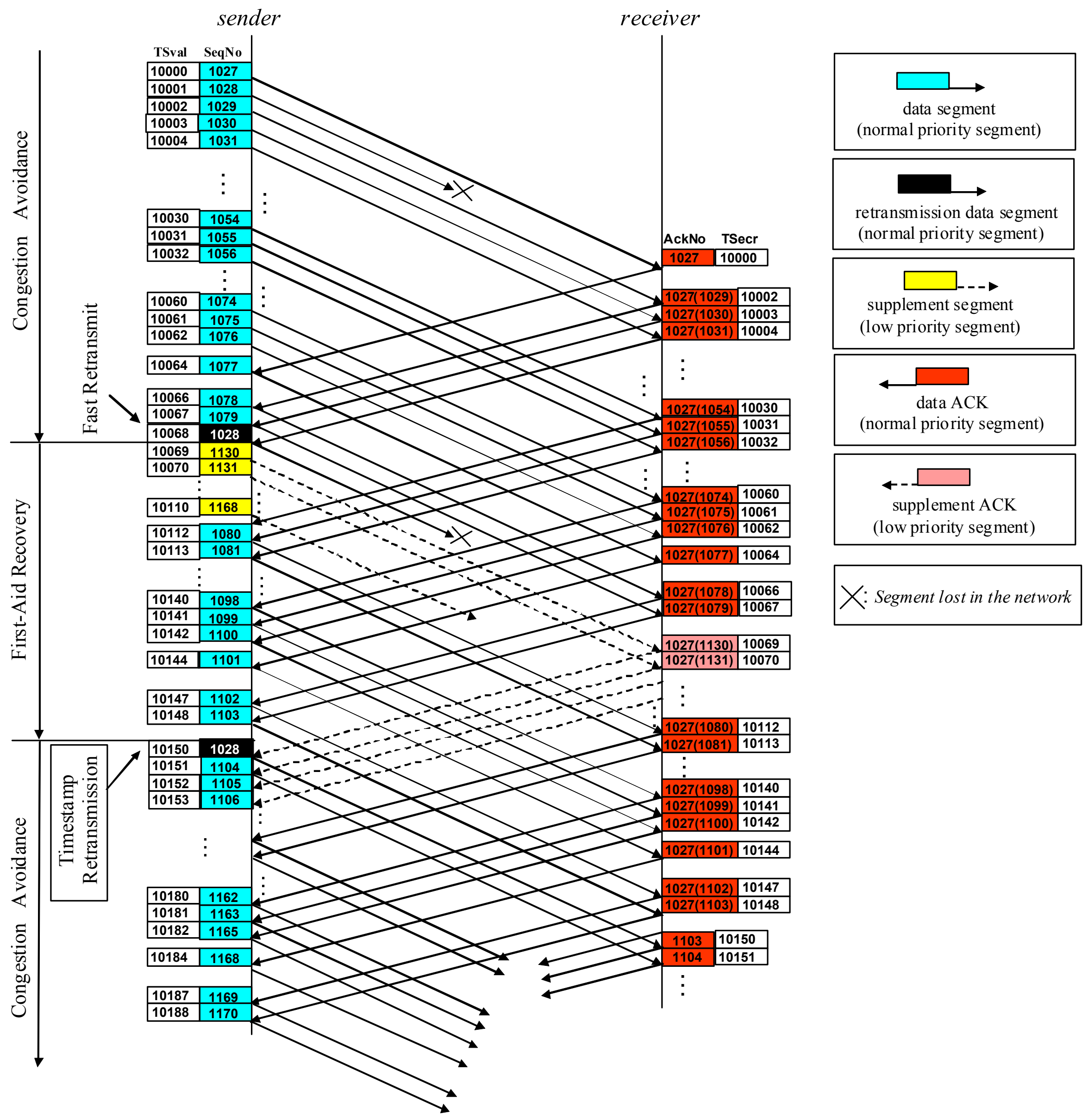

Figure 2. Timestamp retransmission behavior

4. Performance Evaluation

As mentioned in section 1, our objective in this paper is to ensure highest performance from TCP-Cherry in both intra-Planetary and inter-Planetary scenarios using our proposed novel algorithm. For performance evaluation, we use the network simulator, ns-2. As such, we evaluate the performance of TCP-Cherry with Timestamp Retransmission algorithm in deep space links of IPN Internet, through several experiments by varying packet loss probability p. We assume the number of flow N = 1, the link capacity, c = 130 segments/s which is approximately 1Mb/s for TCP segments of 1,000 bytes, the buffer size of the uplink, K = 100 segments, the maximum congestion window size, *maxcwnd* = 76,800 segments, the buffer size of receiver, *rwnd* = 512,000 segments, *RTT* = 600 seconds, and *Connection Duration* = 10,000sec. We vary segment loss probability due to link errors from 10^{-4} to 10^{-2} (/segment) (Akan et al., 2004). Table 1 shows the simulation configuration. Figure 3 and Figure 4 are results from simulations. The definitions of goodput and overhead are the same as (Utsumi et al., 2008). Simulation results show that our proposed algorithm elevates TCP-Cherry performance over IPN significantly (upto 13 times in goodput) particularly under the situation when the link error p = 0.01 (/segment).

Table 1. Simulation configuration 1

Number of Flows	1
Capacity for Deep space link	1 Mbps
RTT of Deep space link	600 sec
Drop rate (random) for Deep space link	10^{-4} – 10^{-2} /packets (varied)
Buffer size for Deep space link	100 packets
Maximum congestion window size	76,800 packets
Buffer size for Receiver	512,000 packets
Segment size	1,000 bytes
IP Version	4
Simulation duration	10,000 sec

Next, we evaluate the performance of TCP-Cherry with Timestamp Retransmission algorithm in deep space links of IPN Internet with congestion, through several experiments by varying packet loss probability p. We assume the number of flows, N = 10, the link capacity, c = 130 segments/s which is approximately 1Mb/s for TCP segments of 1,000 bytes, the buffer size of the uplink, K = 100 segments, the maximum congestion window size, *maxcwnd* = 7,680 segments, the buffer size of receiver, *rwnd* = 512,000 segments, *RTT* = 600 seconds, and *Connection Duration* = 10,000sec. We vary segment loss probability due to link errors from 10^{-4} to 10^{-2}. Table 2 shows the simulation configuration. Figure 5, Figure 6 and Figure 7 are results from simulations.

Table 2. Simulation configuration 2

Number of Flows	10
Capacity for Deep space link	1 Mbps
RTT of Deep space link	600 sec
Drop rate (random) for Deep space link	10^{-4} – 10^{-2} /packets (varied)
Buffer size for Deep space link	100 packets
Maximum congestion window size	76,800 packets
Buffer size for Receiver	512,000 packets
Segment size	1,000 bytes
IP Version	4
Simulation duration	10,000 sec

The definitions of goodput, fairness and overhead are as follows (Utsumi et al., 2008),

$$Goodput = \frac{AmountOfCumulativelyAcknowledgedData}{ConnectionDuration}$$

where *AmountOfCumulativelyAcknowledgedData* is the number of bytes acknowledged at senders (i.e., excluding retransmitted data and SACKed data).

$$Overhead = \frac{AmountOfDuplicate\,\mathrm{Re}\,ceivedData}{AmountOfAll\,\mathrm{Re}\,ceivedData} \times 100(\%)$$

where *AmountOfDuplicateReceivedData* is the number of bytes of duplicate data received (i.e., segments with the same sequence number received more then once) by receivers and *AmountOfAllReceivedData* is the number of bytes of all data received by receivers.

$$Fairness = \frac{\sum_{i=1}^{m} x_i^{\;2}}{m \sum_{i=1}^{m} x_i^2}$$

where *m* connections with same connection durations share the link capacity, and x_i is the goodput of connection i.

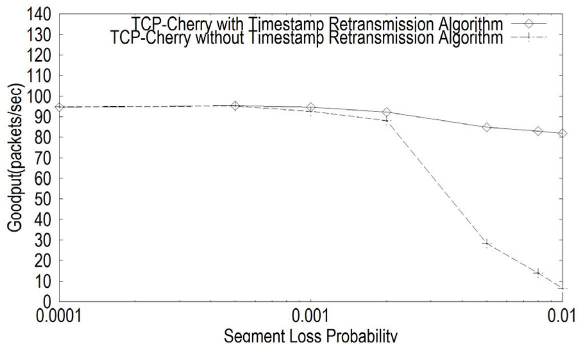

Figure 3. Our algorithm improves goodput significantly at high error rates

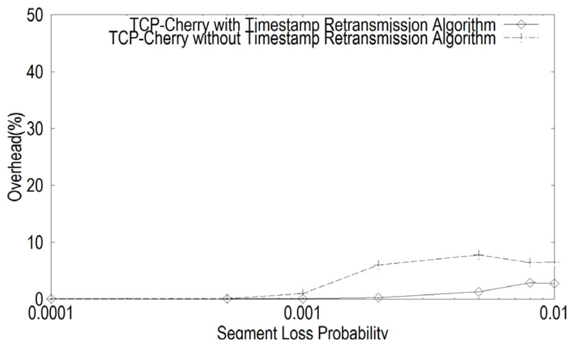

Figure 4. Our algorithm reduces overhead significantly at high error rates

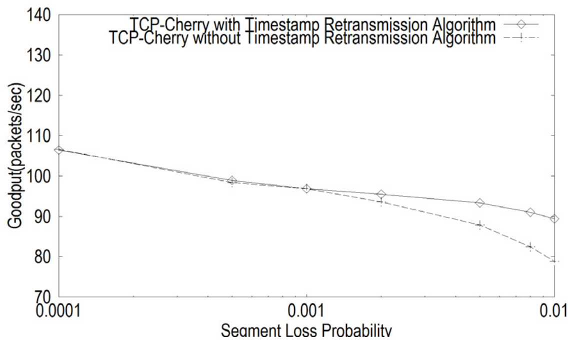

Figure 5. Goodput with 10 flows

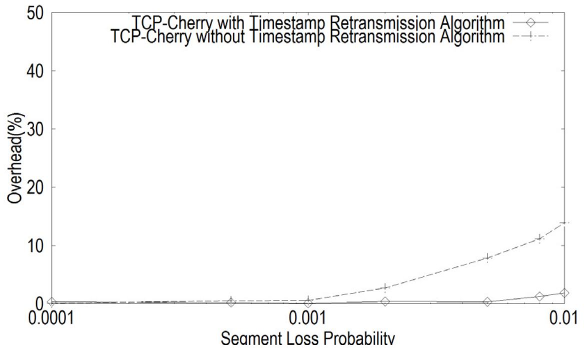

Figure 6. Overhead with 10 flows

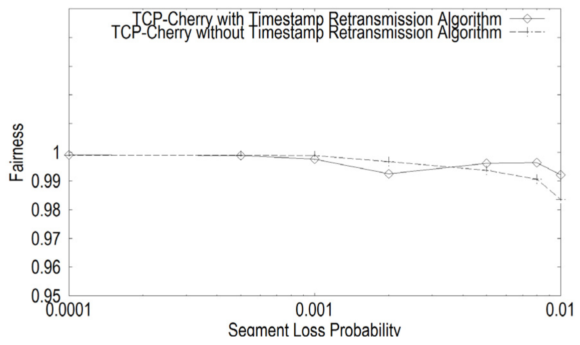

Figure 7. Fairness with 10 flows

Since retransmission algorithms for SCPS-TP and LTP cannot detect lost retransmissions, their performance applied to TCP-Cherry over IPN Internet may be similar to TCP-Cherry without Timestamp Retransmission algorithm, that is, only with the normal fast retransmit algorithm.

5. Conclusion

In this paper we propose a new algorithm, named *Timestamp Retransmission Algorithm*, for deployment along with TCP-Cherry to handle lost retransmissions over IPN Internet efficiently. Simulation results show that our proposed algorithm elevates TCP-Cherry performance over IPN significantly (upto 13 times in goodput) particularly under high link error.

References

Akan, O. B., Fang, J., & Akyildiz, I. F. (2004). TP-Planet: A Reliable Transport Protocol for InterPlaNetary.

Akan, O. B. (2004). TP-planet: a reliable transport protocol for interplanetary Internet. *IEEE Journal on Selected Areas in Communications, 22*(2), 348-361. http://dx.doi.org/10.1109/JSAC.2003.819985

Jacobson, V., Barden, R., & Borman, D. (1992). TCP Extensions for High Performance. RFC 1323.

Kim, B., Kim, D., & Lee, J. (2004). Lost retransmission detection for TCP SACK. *IEEE Commun. Lett., 8*, 600-6002. http://dx.doi.org/10.1109/LCOMM.2004.835326

Kim, B., & Lee, J. (2004). Retransmission loss recovery by duplicate acknowledgement counting. *IEEE Commun. Lett., 8*, 88-90. http://dx.doi.org/10.1109/LCOMM.2003.822525

Utsumi, S., Zabir, S. M. S., & Shiratori, N. (2008). TCP-Cherry: A new approach for TCP congestion control over satellite IP networks. *Computer Communications, 31*, 2541-2561.

Utsumi, S., Zabir, S. M. S., & Shiratori, N. (2009). TCP-Cherry for satellite IP networks: analytical model and performance evaluation. *Computer Communications, 32*, 1377-1383.

Wang, R., Aryasomayajula, N. C., Ayyagari, A., & Zhang, Q. (2007). An Experimental Performance Evaluation of SCPS-TP over Cislunar Communications Links. *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, 2603-2607.

Wang, R., Burleigh, S. C., Parikh, P., Lin, C. J., & Sun, B. (2011). Licklider Transmission Protocol (LTP)-Based DTN for Cislunar Communications. *IEEE/ACM Transactions on Networking, 19*, 359-368. http://dx.doi.org/10.1109/TNET.2010.2060733