A Model for Detecting Transport Layer Data Reneging

Nasif Ekiz, Paul D. Amer
Protocol Engineering Laboratory
Computer and Information Sciences,
University of Delaware

supported by Cisco

PFLDNeT 2010
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
Types of acknowledgements

- For ordered data - cumulative ACK $n$
  - bytes $[\ldots \text{to } n-1]$ (TCP) [RFC 793]
  - segments $[\ldots \text{to } n]$ (SCTP) [RFC 2960]

- For out-of-order data - selective ACK (SACK) $m-n$
  - bytes $[m \text{ to } n-1]$ (TCP) [RFC 2018]
  - segments $[m \text{ to } n]$ (SCTP) [RFC 2960]
  - Prevents unnecessary retransmissions during loss recovery
  - Improves throughput when multiple losses in same window
Receive buffer

Receiving Application

Receive Buffer

- ordered data (ACKed)
- out-of-order data (SACKed)
- available space
Data reneging

- TCP is designed to tolerate reneging
  - [RFC 2018]: “The SACK option is advisory, in that, while it notifies the data sender that the data receiver has received the indicated segments, the data receiver is permitted to later discard data which have been reported in a SACK option.”
  - discarding SACKed data is “reneging”
  - TCP data sender retains copies of all SACKed data until ACKed
TCP and SCTP tolerate reneging

• We argue that tolerating reneging is wrong

1. Hypothesis: “data reneging rarely if ever occurs in practice”
2. Research demonstrates improved performance if SACKed data were not renegable
   ➢ better utilization of send buffer

• Natarajan, Ekiz, Yilmaz, Amer, Iyengar, Stewart, “Non-renegable selective acks (NR-SACKs) for SCTP” Int'l Conf on Network Protocols (ICNP), Orlando, 10/08
   ➢ improved throughput (SCTP only)

• Yilmaz, Ekiz, Natarajan, Amer, Leighton, Baker, Stewart, "Throughput analysis of Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP", Computer Communications. 2010
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
Why study reneging?

- Let’s assume transport protocols are designed to **NOT** tolerate data reneging
  - optimal send buffer utilization
  - improved throughput (SCTP only)

- Changing current TCP and SCTP into non-reneging protocols is easy:
  - SACK semantics changed from advisory to permanent
  - If data receiver needs to renege, data receiver must first **RESET** the connection
Why study reneging?

• Suppose reneging occurs 1 in 100,000 TCP (or SCTP) flows

• Case A (current practice): reneging tolerated
  • 99,999 non-reneging connections underutilize send buffer (and for SCTP may achieve lower throughput)
  • 1 reneging connection continues (maybe?)

• Case B (proposed change): reneging not tolerated
  • 99,999 connections have equal or better send buffer utilization (and for SCTP throughput)
  • 1 reneging connection is RESET
Why study reneging?

- Data reneging has never been studied
  - Does data reneging happen or not?
  - If reneging happens, how often?
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
Detecting reneging at TCP data sender

- TCP has **no** mechanism to detect reneging

- To tolerate reneging, [RFC 2018] suggests the following retransmission policy
  - For each SACKed segment, “SACKed” flag is set
  - “SACKed” segments are not retransmitted until a timeout
  - At timeout, “SACKed” information is cleared
Detecting reneging at SCTP data sender
Detecting reneging at SCTP data sender

Data Sender

Data Receiver

ACK 1

Receive Buffer

1
Detecting reneging at SCTP data sender
Detecting reneging at SCTP data sender

Data Sender

Data Receiver

Receive Buffer

ACK 1

ACK 1, SACK 3-3

ACK 1, SACK 3-4
Detecting reneging at SCTP data sender

Data Sender

1
2
3
4
5

Data Receiver

ACK 1
ACK 1, SACK 3-3
ACK 1, SACK 3-4
ACK 1, SACK 3-5

Receive Buffer

1
3
3
4
3
4
5

Detecting reneging at SCTP data sender

Data Sender → Data Receiver

1.
2.
3.
4.
5.
6.

ACK 1
ACK 1, SACK 3-3
ACK 1, SACK 3-4
ACK 1, SACK 3-5
ACK 1, SACK 3-6

Receive Buffer:

1
3
3 4
3 4 5
3 4 5 6
Detecting reneging at SCTP data sender

Data Sender

Data Receiver

Receive Buffer

1

ACK 1

ACK 1, SACK 3-3

ACK 1, SACK 3-4

ACK 1, SACK 3-5

ACK 1, SACK 3-6

OS needs memory and reneges!
Detecting reneging at SCTP data sender

**Data Sender**

1
2
3
4
5
6

**Data Receiver**

ACK 1

ACK 1, SACK 3-3

ACK 1, SACK 3-4

ACK 1, SACK 3-5

ACK 1, SACK 3-6

OS needs memory and reneges!

**Receive Buffer**

1
3
3 4
3 4 5
3 4 5 6
2

Reneging detected!
Detecting reneging at SCTP data sender

Data Sender

1
2
3
4
5
6
7

Data Receiver

ACK 1
ACK 1, SACK 3-3
ACK 1, SACK 3-4
ACK 1, SACK 3-5
ACK 1, SACK 3-6
OS needs memory and reneges!
ACK 2
ACK 2, SACK 7-7

Receive Buffer

1
3
3 4
3 4 5
3 4 5 6
2
7

reneging detected!
TCP reneging detected at a router

State of receive buffer

Data Sender  Router  Data Receiver

Receive Buffer

1

3
TCP reneging detected at a router

State of receive buffer

Data Sender  Router  Data Receiver

ACK 1, SACK 3-4

Receive Buffer

ACK 1, SACK 3-4
TCP reneging detected at a router

State of receive buffer

Data Sender  Router  Data Receiver

ACK 1, SACK 3-4
ACK 1, SACK 3-6

Receive Buffer

1
3
3 4
3 4 5
3 4 5 6

Data Sender  Router  Data Receiver

1 2 3 4
5 6
TCP reneging detected at a router

State of receive buffer

Data Sender Router Data Receiver

Receive Buffer

ACK 1, SACK 3-4

ACK 1, SACK 3-6

OS needs memory, and reneges!
TCP reneging detected at a router

State of receive buffer

Data Sender -> Router -> Data Receiver

Receive Buffer

- ACK 1, SACK 3-4
- ACK 1, SACK 3-6
- ACK 2, SACK 3-6?

OS needs memory, and reneges!

reneging detected!
Model to detect reneging

- Current state (C) and new SACK (N) are compared
- 4 possibilities:

\[ N \text{ is a superset of } C \quad (N \supseteq C) \]

<table>
<thead>
<tr>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK 12-15</td>
<td>SACK 12-17</td>
</tr>
</tbody>
</table>
Model to detect reneging

- Current state (C) and new SACK (N) are compared
- 4 possibilities:

\[ N \text{ is a superset of } C \ (N \supseteq C) \]
\[ N \text{ is a proper subset of } C \ (N \subset C) \]

<table>
<thead>
<tr>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK 12-15</td>
<td>SACK 12-17</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 12-13</td>
</tr>
</tbody>
</table>
Model to detect reneging

- Current state (C) and new SACK (N) are compared
- 4 possibilities:

<table>
<thead>
<tr>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK 12-15</td>
<td>SACK 12-17</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 12-13</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 22-25</td>
</tr>
</tbody>
</table>

- $N$ is a superset of $C$ ($N \supseteq C$)
- $N$ is a proper subset of $C$ ($N \subset C$)
- $N$ does not intersect with $C$ ($N \cap C = \emptyset$).
Model to detect reneging

- Current state (C) and new SACK (N) are compared
- 4 possibilities:

\[ N \text{ is a superset of } C \ (N \supseteq C) \]
\[ N \text{ is a proper subset of } C \ (N \subset C) \]
\[ N \text{ does not intersect with } C \ (N \cap C = \emptyset) \]
\[ N \text{ intersects with } C, \text{ and } N \text{ and } C \text{ each have some data not in } C \text{ and } N, \text{ respectively} \]
\[ ((N \cap C \neq \emptyset) \land !((N \supseteq C) \land !(N \supset C))) \]

<table>
<thead>
<tr>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK 12-15</td>
<td>SACK 12-17</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 12-13</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 22-25</td>
</tr>
<tr>
<td>SACK 12-17</td>
<td>SACK 15-20</td>
</tr>
</tbody>
</table>
Model to detect reneging

Current state (C)  New SACK (N)  Reneging (R)
Model to detect reneging

CAIDA* trace
- .pcap

TCP flow filter
- tshark
- editcap
- mergecap

Reneg Detect
- ~4600 lines of C code
- ACK reordering check

TCP flows with SACKs
reneging?

*Cooperative Association for Internet Data Analysis
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
Model verification

- RenegDetect was tested with synthetic TCP flows
  - Created reneging flows with text2pcap
  - All reneging flows were identified correctly

- RenegDetect was tested with real TCP flows from CAIDA Internet traces
  - At first, reneging seemed to occur frequently
  - On closer inspection, we found that many SACK implementations are incorrect!

Ekiz, Rahman, Amer, “Misbehaviors in SACK generation” (submitted)
Incorrect SACK implementations

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Misbehavior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>FreeBSD 5.3, 5.4</td>
<td></td>
</tr>
<tr>
<td>Linux 2.2.20 (Debian 3)</td>
<td></td>
</tr>
<tr>
<td>Linux 2.4.18 (Red Hat 8)</td>
<td></td>
</tr>
<tr>
<td>Linux 2.4.22 (Fedora 1)</td>
<td></td>
</tr>
<tr>
<td>Linux 2.6.12 (Ubuntu 5.10)</td>
<td></td>
</tr>
<tr>
<td>Linux 2.6.15 (Ubuntu 6.06)</td>
<td></td>
</tr>
<tr>
<td>Linux 2.6.18 (Debian 4)</td>
<td></td>
</tr>
<tr>
<td>OpenBSD 4.2, 4.5, 4.6, 4.7</td>
<td>Y</td>
</tr>
<tr>
<td>OpenSolaris 2008.05</td>
<td></td>
</tr>
<tr>
<td>OpenSolaris 2009.06</td>
<td></td>
</tr>
<tr>
<td>Solaris 10</td>
<td></td>
</tr>
<tr>
<td>Windows 2000</td>
<td>Y</td>
</tr>
<tr>
<td>Windows XP</td>
<td>Y</td>
</tr>
<tr>
<td>Windows Server 2003</td>
<td>Y</td>
</tr>
<tr>
<td>Windows Vista</td>
<td></td>
</tr>
<tr>
<td>Windows Server 2008</td>
<td></td>
</tr>
<tr>
<td>Windows 7</td>
<td></td>
</tr>
</tbody>
</table>
OUTLINE

1. What is data reneging?
2. Why study reneging?
3. A model to detect reneging
4. Model verification
5. Work in progress
Experiment design – how to “prove” reneging does not happen?

• Event A: TCP flow reneges
• Hypothesis:

\[
H_0: p(A) \geq 10^{-5}
\]

• We want to design an experiment which rejects \( H_0 \) with 95% confidence to conclude

\[
p(A) < 10^{-5}
\]

• Our experiment will observe \( n \) TCP flows hoping to NOT find even a single instance of reneging

\[
\begin{align*}
P(k = 0 \mid H_0) & < .05 \\
p_n(0) & = (1 - 10^{-5})^n \\
(1 - 10^{-5})^n & < 0.05
\end{align*}
\]

• Using MAPLE, \( n \geq 299,572 \)
Questions?
Data reneging in OSes

- Reneging in Linux (version 2.6.28.7)
  - `tcp_prune_ofo_queue()` deletes out-of-order data

- Reneging in FreeBSD, Mac OS
  - `net.inet.tcp.do_tcpdrain` `sysctl` turns reneging on/off
  - `tcp_drain()` deletes out-of-order data
Data reneging in Linux

/*
 * Purge the out-of-order queue.
 * Return true if queue was pruned.
 */
static int tcp_prune_ofo_queue(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    int res = 0;

    if (!skb_queue_empty(&tp->out_of_order_queue)) {
        NET_INC_STATS_BH(sock_net(sk), LINUX_MIB_OFOPRUNED);
        skb_queue_purge(&tp->out_of_order_queue);

        /* Reset SACK state. A conforming SACK implementation will
         * do the same at a timeout based retransmit. When a connection
         * is in a sad state like this, we care only about integrity
         * of the connection not performance.
         */
        if (tp->rx_opt.sack_ok)
            tcp_sack_reset(&tp->rx_opt);
        sk_mem_reclaim(sk);
        res = 1;
    }
    return res;
}
3. Inferring the state of receive buffer

<table>
<thead>
<tr>
<th>TCP Segments with n SACK options</th>
<th>Enough space for another SACK option</th>
<th>Not enough space for another SACK option</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=1</td>
<td>~88%</td>
<td>0%</td>
</tr>
<tr>
<td>n=2</td>
<td>~11%</td>
<td>0%</td>
</tr>
<tr>
<td>n=3</td>
<td>0.7%</td>
<td>0.20%</td>
</tr>
<tr>
<td>n=4</td>
<td>n/a</td>
<td>0.15%</td>
</tr>
<tr>
<td>Total number of TCP segments</td>
<td></td>
<td>780,798 (100%)</td>
</tr>
</tbody>
</table>
3. Inferring the state of receive buffer

<table>
<thead>
<tr>
<th>TCP Segments with n SACK options</th>
<th>Enough space for another SACK option</th>
<th>Not enough space for another SACK option</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=1</td>
<td>~88%</td>
<td>0%</td>
</tr>
<tr>
<td>n=2</td>
<td>~11%</td>
<td>0%</td>
</tr>
<tr>
<td>n=3</td>
<td>0.7%</td>
<td>0.20%</td>
</tr>
<tr>
<td>n=4</td>
<td>n/a</td>
<td>0.15%</td>
</tr>
<tr>
<td>Total number of TCP segments</td>
<td></td>
<td>780,798 (100%)</td>
</tr>
</tbody>
</table>
Misbehaviors in SACK generation

- 7 misbehaviors are observed in CAIDA traces
- We designed TBIT tests to verify SACK generation
- 27 OSes are tested
- RenegDetect is updated to identify those misbehaviors
Example TBIT test

1. SYN (SACK-Permitted, ISN 400)
2. SYN-ACK (SACK-Permitted)
3. SEQ 401-402 (1)
4. ACK 402
5. SEQ 403-404 (1)
6. ACK 402, SACK 403-404
7. SEQ 405-406 (1)
8. ACK 402, SACK 405-406, 403-404
9. SEQ 407-408 (1)
10. ACK 402, SACK 407-408, 405-406, 403-404
11. SEQ 409-410 (1)
12. ACK 402, SACK 409-410, 407-408, 405-406, 403-404
13. RST

Misbehavior A1: Missing Y
Misbehavior A2: Missing X and Y