TCP Congestion Control With a Misbehaving Receiver

Stefan Savage, Neal Cardwell, David Wetherall and Tom Anderson

- Pavan Kumar Panjam
Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet, performance degrades.

- If every user sends data at a very high rate, it causes congestion.
  - Packets will be dropped (Unreliable transmission)
- If every user sends data at a very low rate, resource will not be well utilized.

- Users need to send data at a correct rate so that
  - Resources are well utilized and gets reliable data transfer.
  - Resources should be shared fairly.

This is the primary goal of Congestion Control.
A Brief Overview of TCP Error and Congestion Control Mechanisms

- Sender → data → segments (SMSS) labeled with seq No:
- Receiver → ACK (Cumulative Acknowledgement)
- TCP uses several algorithms for congestion control
  - Slow Start
  - Congestion avoidance
    These algorithms control the sending rate by manipulating a congestion window (cwnd).
- \textit{Cwnd} limits the number of outstanding unacknowledged bytes that are allowed at any time.
• When a connection starts
  – Slow start algorithm: increases $cwnd$ quickly till it reaches bottleneck.
• When the sender infers the segment has been lost, it interprets this as an implicit signal of network overload and decreases $cwnd$ quickly.
• After approx. bottleneck capacity, TCP switches to Congestion Avoidance algorithm.
• Congestion Avoidance: increases $cwnd$ more slowly to probe for additional bandwidth that may become available.
Now, Exploit the vulnerabilities

Attack 1: ACK division

- Upon receiving a data segment containing N bytes, the receiver divides the resulting acknowledgment into M, where M <= N, separate acknowledgments – each covering one of M distinct pieces of the received data segment.
TCP uses two algorithms 'fast retransmit' and 'fast Recovery' to mitigate the effects of packet loss.

- Fast Retransmit: Detects loss by observing three duplicate acknowledgments and it immediately retransmits what appears to be the missing segment.

- The fast Recovery algorithms employs this information as follows:(quoted from RFC 2581).
  - Set cwnd to ssthresh plus 3*SMSS. This artificially “inflates” the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.
Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received (at the start of a connection this would be for the SYN segment).
Attack 3: Optimistic ACKing

- Upon receiving a data segment, the receiver sends a stream of acknowledgments anticipating data that will be sent by the sender.
Implementing these Attacks (Proof)

- Modified the TCP subsystem of linux 2.2.10 for this purpose.
- The resulting TCP implementation provides extremely high performance at the expense of its competitors.
- The pictures shown here are time sequence plots of packet traces for both normal and modified receiver TCP's.
DupACK spoofing
Optimistic Aeking
### Applicability (Tested on different servers)

<table>
<thead>
<tr>
<th>OS Name</th>
<th>ACK Division</th>
<th>DupACK Spoofing</th>
<th>Optimistic Acks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris 2.6</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Linux 2.0</td>
<td>Y</td>
<td>Y (N)</td>
<td>Y</td>
</tr>
<tr>
<td>Linux 2.2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Windows NT4/95</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>FreeBSD 3.0</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DIGITAL Unix 4.0</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IRIX 6.x</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HP-UX 10.20</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>AIX 4.2</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Solutions

- **Design Robust Protocols:**
  - TCP's vulnerabilities arise from a combination of unstated assumptions, casual specification and a pragmatic need to develop congestion control mechanisms that are backward compatible with previous TCP implementations.

- Abadi and Needham's paper, *Prudent Engineering Practice for Cryptographic Protocols*, presents a set of design rules that are surprisingly germane to this problem [AN96].

- **First three principles of it are stated below (pertinent to this problem)**
  - Principle 1: Every message should say what it means: the interpretation of the message should depend only on its content.
  - Principle 2: The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.
  - Principle 3: If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.
Solution to Attack 1: ACK Division

- This vulnerability arises from an ambiguity about how ACKs should be interpreted – a violation of the second principle.

- Two obvious solutions:
  - Either modify the congestion control mechanisms to operate at byte granularity or guarantee that segment-level granularity is always respected.
  - Increment $cwnd$ by one SMSS when a valid ACK arrives that covers the entire data segment sent. (This technique is employed in the latest versions of Linux (2.2.x)).
Solution to Attack 2: DupACK spoofing

- During fast recovery and fast retransmit, TCP's design violates the first principle – the meaning of a duplicate ACK is implicit, dependent on previous context, and consequently difficult to verify.

- **Singular Nonce:**
  - Introduce two new fields into the TCP packet format: Nonce and Nonce reply. For each segment, the sender fills the Nonce field with a unique random number generated when the segment is sent. When a receiver generates an ACK in response to a data segment, it echoes the nonce value by writing it into the Nonce Reply field.
Solution to Attack 3: Optimistic ACKing

- The optimistic ACK attack is possible because ACKs do not contain any proof regarding the identity of the data segment(s) that caused them to be sent.

- In the context of the third principle described earlier, a data segment is a principal and an ACK is the message of concern.

- **Cumulative Nonce:**
  - For each segment, the sender fills the Nonce field with a unique random number generated when the segment is sent. Each side maintains a nonce sum representing the cumulative sum of all in-sequence acknowledged nonces. When a receiver receives an in-sequence segment it adds the value contained in its Nonce field to this sum. When a receiver generates an ACK in response to a data segment, it either echoes the current value of the nonce sum (for in-sequence data) or echoes the nonce value sent by the sender (for out-of-sequence data).
Conclusion

- 'Security' only provides low-level primitives
  - Authentication, Privacy, Access Control etc...
- Q: Is other end-point sending semantically correct data?
  - Rational Incentives
  - Malice
  - Bugs
- Q: Can you verify the semantics of the message?