Matt Anderson and Lawrence Xing
andersme, lxing

TCP Congestion Control with a Misbehaving Receiver

Introduction

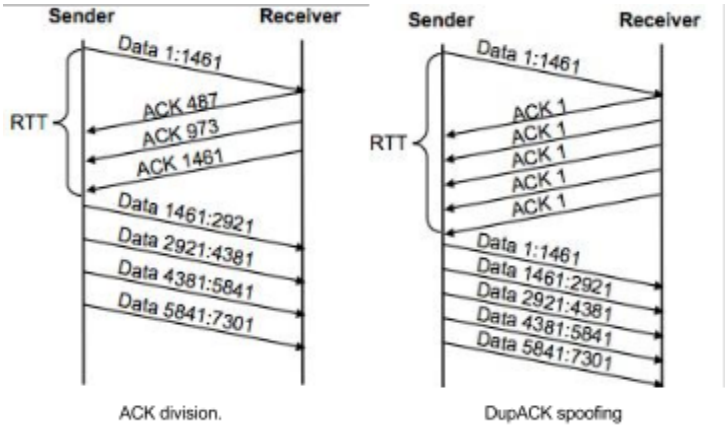
At the time when many of the Internet's protocols were being designed and implemented, the Internet was a small and cooperative place, comprised primarily of hosts managed by engineers that were united by the common cause of connecting computers effectively, reliably, and fairly. Among these protocols was TCP, still one of the most used on the Internet today. Although it has changed some over the years, responding to the challenges of Internet congestion and demands for ever-higher transfer rates, TCP remains dependent on the cooperation of the sender and receiver to ensure, among other things, a fair distribution of throughput between flows competing for the same bottleneck link.

In "TCP Congestion Control with a Misbehaving Receiver," Savage et al. demonstrate three attacks, each of which a greedy receiver might employ to induce a sender to transmit data arbitrarily fast. In real terms, a user could deploy these attacks on their personal computer's TCP stack and expect to load webpages (especially small ones) much faster. This would create an incentive for many users to do the same, a deeply problematic outcome, since the congestion control algorithms these attacks exploit are key to preventing Internet-wide congestion collapse. In this blog post, we attempt to reproduce two of Savage et al.'s attacks, termed ACK division and DupACK spoofing.

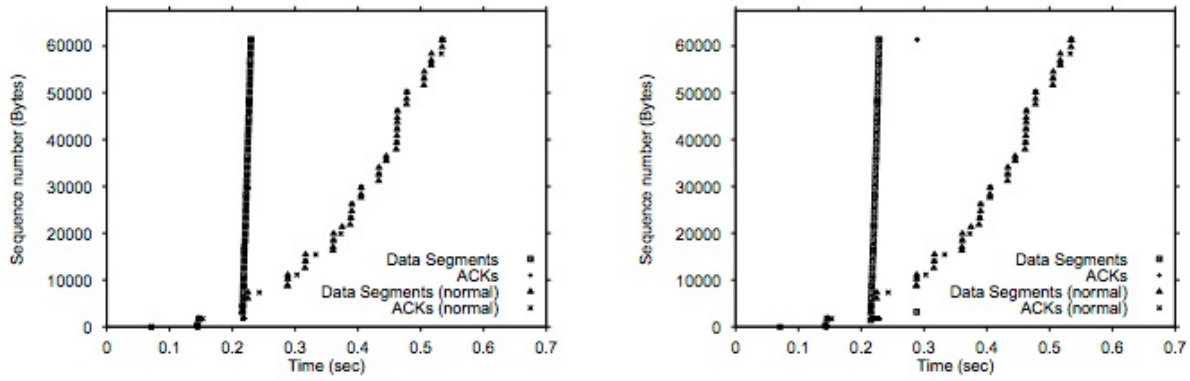
The Attacks

ACK division exploits the incongruity between TCP's error control protocol, implemented with byte-granular *seqnos* and *acknos*, and its congestion control protocol, which is defined in terms of segments. According to RFC 2581, "[d]uring slow start, TCP increments *cwnd* by at most [one segment] for each ACK received that acknowledges new data." Assuming slow start begins by sending a *cwnd* of one segment, then a cooperative receiver responds by sending a single, cumulative ACK, causing *cwnd* to double. Under ACK division, however, the receiver sends M separate ACKs (each acknowledging 1/M of the initial *cwnd*'s bytes), which causes the sender to inflate its *cwnd* much faster.

DupACK spoofing, on the other hand, exploits the fast retransmit and fast recovery features of TCP congestion control. Upon receipt on a triple duplicate ACK, a typical sender retransmits unacknowledged data immediately. According to RFC 2581, the sender should also "[s]et *cwnd* to *ssthresh* plus 3*[sender's maximum segment size]" and "increment *cwnd* by [the sender's maximum segment size]" for each additional duplicate ACK received. This behavior is readily exploited by DupACK spoofing, which deliberately sends multiple copies of the same ACK to induce a flood of traffic from the sender. Both attacks are illustrated below.



When implemented as modifications to the TCP stack in the Linux 2.2 kernel and used to request a webpage, Savage et al. produced the following plots, which show remarkable gains in transfer rate (ACK division is on the left, DupACK spoofing on the right):



Methodology:

Details forthcoming.

Challenges, Analysis, and Criticism:

Details forthcoming.

Progress and Future Plans:

Thus far, Lawrence and I have been working to develop a testbed for trying out the three algorithms. We were initially optimistic about modifying the Linux kernel's TCP stack but, faced with daunting compilation and debugging times, ultimately settled on LwIP, a lightweight TCP/IP stack that operates in user space. Since then, we've hacked an example LwIP TCP echo server into acting as an iperf server and implemented rudimentary ACK division in the LwIP TCP stack. Initial hand-testing over the loopback interface shows that (1) ACK division has introduced considerable instability (which is to be expected) in the flow between the iperf

client and our hacked iperf server (at least when the RTT is very low) and (2) that our hacked iperf server might be limited to 300 MBits/sec by weaknesses in the implementation. In the coming weeks we will iterate on the iperf server, develop a bona fide Mininet topology and test harness, and implement DupACK spoofing.