Eliminating Aborted Data Delivery Over Cellular Links (Poster Abstract)

Andrei Gurtov
gurtov@cs.helsinki.fi
University of Helsinki, Finland and ICSI, Berkeley, USA

Internet users often abort their transfers in progress, for example by clicking on the "Reload" button or another link in a web browser. Analysis of backbone Internet traces [5] shows that 15-30% of all TCP connections are abnormally terminated via a reset. The average length of reset connections is not significantly different from completed connections. Thus, resets are not merely generated by TCP connection refusals or misbehaving firewalls [1].

Packets from aborted transport connections are often sent unnecessarily to the user over a slow last-hop link delaying useful traffic. This is a particular concern for wide-area wireless links, because unnecessary transmissions waste scarce radio bandwidth, battery power at the mobile terminal and incur monetary cost due to billing by data volume.

Although the total fraction on aborted traffic is only a few percent in backbone Internet traces, we believe it is a significant problem for slow last-hop links for two reasons. First, users easily get impatient while waiting for a transfer to complete over a slow link. Second, the round trip time for such links can reach several seconds during a bulk transfer. This delays delivery of an abort notification from the receiver to the sender.

Measurements in a live cellular network show that loading of a new web page is often delayed up to ten seconds due to delivery of data from previously aborted pages. Figure 1 shows a trace of a web page download being aborted. It takes seven seconds until the server receives the abort notification, stops sending and page data drains from the link queue. About 25 kilobytes of data is unnecessarily transmitted over a 30-kbps GPRS link.

Deploying an active queue management algorithm, such as RED, in the last-hop router can reduce the number of aborted packets sent over the last-hop link. AQM keeps the average queue size low without penalizing bursty sources. However, cellular links require a buffer larger than the bandwidth-delay product of the link to efficiently implement local error recovery. In existing cellular networks we often find the bottleneck buffer of about ten times the bandwidth-delay product of the link [3].

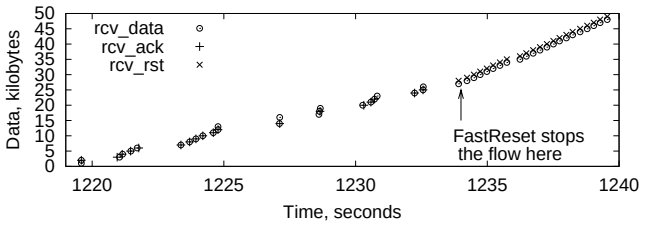

Figure 1: Aborted TCP connection over GPRS to www.zed.com (receiver trace, Mozilla/HTTP 1.1).

The standard TCP receiver generates reset (RST) packets after receiving (and discarding) packets on an aborted connection. We propose a Fast Reset algorithm to eliminate delivery of aborted data over a last-hop link. The algorithm can be included into a performance enhancing proxy snooping packet headers at the last-hop link. The algorithm is illustrated in Figure 2 and has the following steps:

1. An application on the mobile client opens a TCP connection and requests a web object.
2. The server receives the request and starts transmitting data to the client.
3. Data packets are buffered in the last-hop router and transmitted to the client.
4. The user decides to abort the download, for example by pressing a "Reload" button. The TCP receiver responds with reset packets to data packets arriving from the server.
5. The last-hop router notices a reset packet and discards buffered packets in the downlink direction that belong to the aborted connection. It then forwards the reset packet toward the server.
6. The server receives the reset packet and stops transmitting data on the aborted connection.

Step 5 can be further detailed as follows:

```
for every arriving pkt1 in uplink
if pkt1 is RST
  for each queued pkt2 in downlink
    if (src1, dst1, sport1, dport1, prot1)
    == (dst2, src2, dport2, sport2, prot2)
                discard(pkt2)
forward(pkt1)
```

The 5-tuple (src, dst, sport, dport, prot) uniquely identifies a transport connection by its source and destination IP addresses, source and destination port numbers and the protocol number.

If a reset packet gets lost, the sender retransmits a TCP segment after a timeout. This segment will not be dropped by the last-hop router according to the algorithm. Instead, the retransmitted segment will generate another reset packet at the receiver. There is no state kept in the router, thus there is no harm to new TCP connections if the client reuses ports from aborted connections. Fast Reset works robustly for all TCP applications including HTTP, FTP, and peer-to-peer.

Since web browsing is the main contributor of aborted connections, we describe how resets affect HTTP in detail. HTTP v1.0 allows a maximum of four concurrent TCP connections to a server. Thus, aborting a web page download can generate resets for several TCP connections. This reduces the performance benefit of Fast Reset, because aborted data contained in the buffer of the last-hop router is distributed over several TCP connections.

HTTP v1.1 defines persistent connections. With pipelining, several HTTP requests can be outstanding over a single TCP connection. Once an abort of a single request is initiated, the whole pipeline is aborted. This helps the Fast Reset algorithm, because with a single reset packet unnecessary data from several data objects can be eliminated.

We evaluated Fast Reset using the Netscape browser in Linux over a GPRS cellular link. When the loading of a web page is aborted, 5-25 packets unnecessarily arrive to the receiver from aborted TCP connections. This corresponds to 1-10 seconds of time wasted before a new page begins loading. Fast Reset reduces the penalty of aborting a web page to one-two packets per a TCP connection. As a result, the response time is reduced, battery power and radio spectrum are preserved.

The main limitation of the Fast Reset algorithm is the requirement for a TCP connection to traverse through the last-hop router in both directions. Although we expect this requirement to be met in most cases, there are asymmetric setups, e.g. for satellite links [6]. Another drawback of Fast Reset is the processing load on the router to parse the queue on every arriving reset packet. We expect the load to be acceptable for low rate links [2]. Finally, Fast Reset creates a layering violation because the router has to access transport-layer headers. TCP/IP header compression is another example of a layering violation. If the network-layer protocol, such as ISO CONP, is connection-oriented, Fast Reset can avoid this problem.

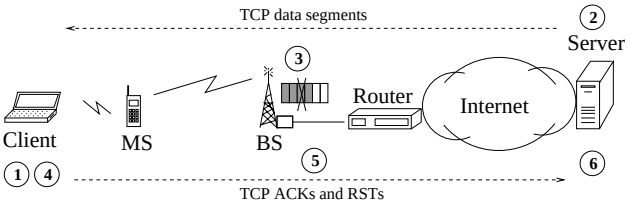

Figure 2: The Fast Reset algorithm.

Fast Reset can also be implemented in the mobile client to eliminate aborted data delivery for uplink transfers. We expect it to have only a modest performance gain because the clients and not servers seem to initiate aborts. Fast Reset can be adopted for other connection-oriented transport protocols than TCP, such as SCTP and DCCP [4].

References

[1] S. Floyd. Inappropriate TCP resets considered harmful. IETF RFC 3360, Aug. 2003.

[2] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. of ACM SIGCOMM'99*, Aug. 1999.

[3] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola. Multi-layer protocol tracing in a GPRS network. In *Proc. of IEEE Vehicular Technology Conference (VTC'02 Fall)*, Sept. 2002.

[4] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion control without reliability. Available at http://www.icir.org/kohler/dccp/, May 2003.

[5] MAWI. Packet traces from WIDE backbone. Available at http://tracer.csl.sony.co.jp/mawi/, Aug. 2003.

[6] V. Padmanabhan, H. Balakrishnan, K. Sklower, E. Amir, and R. Katz. Networking using direct broadcast satellite. In *Proc. of Workshop on Satellite-Based Information Systems*, Nov. 1996.