BIBLIOGRAPHY

[BCS94] R. Braden, D. Clark and S. Shenker, “Integrated Services in the Internet Architecture: an Overview”, *Internet RFC 1633*, June 1994.
http://sunsite.auc.dk/RFC/rfc/rfc1633.html

[Bla98] S. Blake et al., "An Architecture for Differentiated Services", *Internet RFC 2475*, December 1998.
http://sunsite.auc.dk/RFC/rfc/rfc2475.html

[CF98] D. Clark and W. Fang, “Explicit Allocation of Best-Effort Packet Delivery Service”, IEEE/ACM Transactions on Networking, Vol. 6, N. 4, pp. 362-373, August 1998.
http://www.cs.princeton.edu/~wfang/Research/expected-cap.ps

[Fer2000] A. Feroz, “A TCP-Friendly Traffic Marker for IP Differentiated Services”, Master Thesis, Rensselaer Polytechnic Institute, February 2000.

[Fer1999] A. Feroz et al., “TCP-Friendly Traffic Conditioners for Differentiated Services”, Internet Draft, *draft-azeem-tcpfriendly-diffserv-00.txt*, March 1999.
http://networks.ecse.rpi.edu/~amit/diffserv/tcpfriendly.ps

[FF96] K. Fall and S. Floyd, “Comparison of Tahoe, Reno and Sack TCP”, Computer Communication Review, July 1996.
ftp://ftp.ee.lbl.gov/papers/sacks_v2.ps.Z

[FJ93] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance”, *IEEE/ACM Transactions on Networking*, Vol. 1, N. 4, pp. 397-413, August 1993.
http://www.aciri.org/floyd/papers/red/red.html

[Jac88] V. Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM’88, pp.314-329, August 1988.
ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

[LM97] D. Lin and R. Morris, “Dynamics of random early detection”, *Proceedings of SIGCOMM’97*, 1997.
http://www.pdos.lcs.mit.edu/~rtm/papers/fred.ps

[Man98] T. Mancill, “Linux WAN Routers. Musings of a Network *Linux Journal*, N. 50, June 1998.
http://www2.linuxjournal.com/lj-issues/issue50/2928.html

[May99] M. May, “Reasons not to deploy RED”, Technical Report, June 1999.
http://www-sop.inria.fr/rodeo/personnel/mmay/may_red.html

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, “TCP Selective Acknowledgment Options”, *Internet RFC 2018*, October 1996.
http://www.aciri.org/floyd/papers/rfc2018.txt

[Sah99] S. Sahu et al., “On Achievable Service Differentiation with Token Bucket Marking for TCP”, UMASS CMPSCI Technical Report, pp. 99-72, November 1999.
ftp://gaia.cs.umass.edu/pub/Sahu99-tcp-tb.ps.gz

[Ste95] W. Stevens, *TCP/IP Illustrated*, Vol. I, Addison-Wesley, Reading, MA, 1995.

[Ste97] W. Stevens, TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms”, *Internet RFC 2001*, January 1997.
http://sunsite.auc.dk/RFC/rfc/rfc2001.html

[WHK99] W. Almesberger, J. Hadi and A. Kusnetsov, “Differentiated Services on Linux", Internet Draft, *draft-almesberger-wajhak-diffserv-linux-01.txt*, June 1999.
ftp://lrcftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt

APPENDIX

Round Trip Time Delay Component

A.1 Motivations

The experimentation process carried out in our Linux testbed lab, did not allow us to perceive the performance differences between TCP Reno and TCP SACK. We recall that one of the greatest benefits offered by SACK relies on the fact that successfully received segments are never retransmitted by the source, under any circumstances. This is not the case for Reno, for which, on the event of a timeout, the pipe is essentially flushed out and all segments, since the last one acknowledged, have to be retransmitted. We intuitively understand that the longer the link communicating the source and the destination, the better the performance achieved by SACK. Incidentally, for short links (with small round trip time delay), the performance seen with SACK does not significantly change from that with Reno.

In addition, we recall that TCP sets up its retransmission timers on the basis of the round trip time for a connection. The calculation of this round trip time is done as often as a time-stamped ack is received by the source. This resulted on a greatly unstable round trip time estimate, since at some instances the congestion was much worse than at some others. This is exacerbated by the fact that the round trip times are usually very small (between 100 and 500 microseconds) but some other times much longer. The final effect of this behavior was a large number of spurious timeouts that did not correspond with the reality outside our experimentation environment. We implemented this delay component in the network, in order to better emulate the real behavior of our system on the backbone of the Internet.

A.2 Delay Component Implementation

We followed a similar basic approach as the one for realizing the buffer on the TCP Friendly marker and the FRIO kernel implementations. We place a queue on the outgoing direction of the destination (D) interface, and hold the packets for the time specified on installation of the component. This results in effectively holding the acks that leave the destination as to emulate that the round trip transfer of data has in fact taken such amount of time. We note that neither the source nor the destination ever know that the time of the transfer is in fact the time the data is being held at the queue, but instead assume that the link is a much longer one.

When we enqueue a packet, we record its arrival time into the queue. Later, when the packet makes it to the head of the queue, we only dequeue it if the proper time has elapsed. We append instances of a our own time data structure into a list, as packets go in the queue and keep a one to one correspondence between the position of these structures, in the list, and the position of the packets they belong to, in the buffer. Finally, we note that it is critical to have a large enough buffer size, with size directly proportional to the delaying time, to prevent this scheme from loosing packets as it runs out of memory to hold them.