

White Paper
**TCP and Queue
Management**



Agilent N2X



Agilent Technologies

Background and Motivation

Transmission Control Protocol (TCP) and various queue management algorithms such as tail drop and random early detection (RED), etc. are intimately related topics. Historically, the two evolved together and have had a symbiotic relationship. However, a poor understanding of the relationship between TCP and queue-management algorithms is at the root of why measured performance in routers is dramatically different from the actual performance of live networks, leading to a need for massive over-provisioning. Explanations of how the various components of TCP work, and the effect various queuing schemes have on TCP, tends to be isolated on specific topics and scattered. This paper attempts to bring the key concepts together in one place with the objective of improving understanding of these two critically linked topics.

Transmission Control Protocol

TCP is a layer-4 protocol in the 7-layer Open Systems Interconnection (OSI) model. TCP sits on top of the Internet Protocol (IP), and is used by applications to communicate between servers and clients. TCP is a connection-oriented and robust transmission algorithm, in that it establishes a connection between client and server before sending data, and in that it acknowledges all the data that is sent, re-transmitting any data lost along the way. TCP is commonly used for applications such as mail (IMAP, SMTP, POP3) file transfer (FTP), file sharing (peer-to-peer), web-surfing (HTTP), music downloads (iTunes) and video downloads (VoD, YouTube). TCP's connectionless and less-robust counterpart, UDP, is often co-existing on the same networks and links as UDP, and is used for streamed applications such as VoIP and IPTV, where received data must be decoded and displayed immediately, and re-transmissions cannot occur fast enough to meet minimum delay requirements. UDP is also used for applications taking advantage of multicast.

TCP Incarnations

Early TCP – In the beginning of the Internet, i.e. ArpaNET, TCP was very simple and would start by transmitting a whole window's worth of data. Under congestion conditions which started to emerge around 1986, intermediate routers were forced to discard many packets. TCP, as it existed at the time, would actually double the rate at which packets were sent, pushing the entire network into a state of congestion collapse, where the network effectively ground to a halt, with the operational capacity of networks dropping by a factor of one thousand'.

Slow start – The slow start mechanism was a direct attempt to avoid congestion collapse by increasing the packet rate of a connection in a controlled fashion – slowly at first, faster later on - until congestion is detected (i.e. packets are dropped), and ultimately arrive at a steady packet rate, or equilibrium. To achieve this goal, the designers of the slow start mechanism chose an exponential ramp-up function to successively increase the window size. Slow start introduces a new parameter called the congestion window or *cwnd*, which specifies the number of packets that can be sent without needing a response from the server. TCP starts off slowly (hence the name "slow start") by sending just one packet, then waits for a response (an ACK) from the receiver. These ACKs confirm that it is safe to send more packets into the network (i.e. double the window size), rather than wait for a response packet by packet. The window size grows exponentially until a router in between the sender and the receiver discards (drops) some packets, effectively telling TCP through a time-out event that its window size has grown too large. Figure 1 illustrates how the TCP window size is adjusted dynamically over time with the slow-start mechanism. Slow start is usually characterized as a "congestion control" mechanism, and is commonly implemented in modern implementations of TCP.

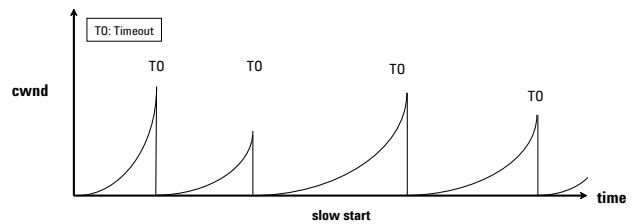


Figure 1: TCP congestion window dynamics with slow start

Congestion Avoidance – When a network is congested, queue lengths start to increase exponentially. Congestion avoidance was devised as a technique to signal packet loss via time-outs and make TCP throttle back quickly (more quickly than queue lengths are growing), with the objective of stabilizing the whole system. Near the point of congestion, overly aggressive increases in connection bandwidth can drive the network into saturation, causing the "rush-hour effect". To avoid the rush-hour phenomenon, the congestion avoidance mechanism increases bandwidth additively rather than exponentially. When congestion is detected via a timeout, bandwidth is scaled back aggressively by setting *cwnd* to half the current window size. Congestion avoidance is sometimes characterized as being an additive-increase, multiplicative-decrease (AIMD) technique.

While slow start and congestion avoidance were devised separately and for different purposes, they are almost always implemented together. The combined algorithm introduces a new variable called *ssthresh* (slow-start threshold), that effectively determines which mechanism is used. As shown in Figure 2, if *cwnd* is less than or equal to *ssthresh*, then TCP is in slow start. If *cwnd* is greater than *ssthresh*, then TCP is in congestion avoidance.

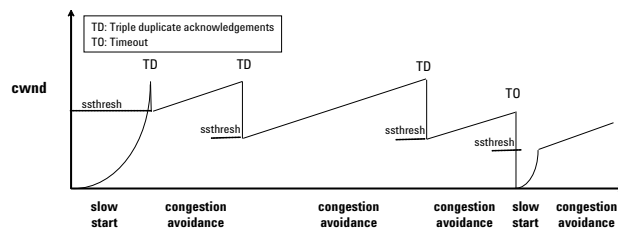


Figure 2: TCP congestion window dynamics with slow start and congestion avoidance

Fast Retransmit – Early TCP was designed with a simple retransmission timer that was very coarse. If packets were lost due to network congestion, considerable idle time was wasted before TCP would start transmitting again. Fast retransmit is an attempt to accelerate TCP’s responsiveness through the use of ACK packets. As a TCP receiver receives each packet, it is continually sending ACKs back to a TCP sender. A TCP receiver sends back duplicate packets when a packet is lost, or packets are received in the wrong order. If three duplicate ACKs are sent back from receiver to sender, an assumption is made that packets have been lost in the network and they must be re-transmitted. A TCP sender then re-transmits the segment that has been lost without waiting for a re-transmission timer to expire.

TCP Tahoe – When the first three of these techniques (slow start, congestion avoidance, fast retransmit) are used together, that implementation is nicknamed “Tahoe”.

Fast Recovery – Fast recovery works hand in hand with fast retransmit to improve the responsiveness of TCP once congestion has occurred. Fast recovery introduces the concept of partial acknowledgements, which are used to notify that the next in-sequence packet has been lost so it can be re-transmitted immediately. Furthermore, in stead of going back into slow-start mode, fast recovery jumps TCP into congestion avoidance mode. The congestion window is also reduced to the slow start threshold (*ssthresh*) instead of dropping all the way back to a window size of one packet.

The four techniques above are described in IETF RFC 2581².

TCP Reno – When the first four of these techniques (slow start, congestion avoidance, fast retransmit and fast recovery) are used together, that implementation is nicknamed “TCP Reno”.

Modified Fast Recovery – Engineers noticed that packets tended to be dropped from queues in bursts, and sought to improve TCP Reno’s performance in that situation, i.e. specifically when multiple packets were dropped from the same window. Modified fast recovery introduces a new variable called *recover* and is detailed in RFC 3782³.

TCP New Reno – When TCP uses the modified fast recovery scheme along with the four congestion techniques above, that implementation is nicknamed “TCP New Reno”.

Selective Acknowledgement – As data traffic grew in networks, TCP was used to handle larger and larger window sizes (the effective capacity of a TCP connection is a function of the largest window size). The larger the window size, the more vulnerable TCP’s performance is to dropped packets, where multiple packets lost from the same window cause a catastrophic effect on TCP’s performance. Selective Acknowledgement, or SACK, is an attempt to decrease that vulnerability and improve TCP’s performance in congested networks. As mentioned above, a TCP receiver sends ACKs back to the TCP sender continuously as data packets are received. Before selective acknowledgement was devised, TCP could only acknowledge the latest sequence number of received contiguous data. Select Acknowledgement, or SACK, allows the receiver to acknowledge non-contiguous blocks of received data. The next effect is that with SACK, TCP becomes more robust to packet loss. SACK is common on modern networks because it is employed in Windows 2000. IETF RFC 2018⁴, published in 1996, defines selective acknowledgement.

TCP Vegas – All of the congestion control mechanisms described above are reactive in that they respond to packet loss in various ways. TCP Vegas is an attempt to be proactive, predict at what level congestion occurs, and avoid buffers from overflowing. TCP Vegas calculates the actual sending rate (the flight size divided by the round-trip time) and compares the result with the expected throughput (window size divided by the base round-trip time). TCP Vegas then makes incremental or decremental adjustments to the congestion window. Small decreases in window size are acceptable because they are made before congestion is detected. If actual congestion occurs, TCP Vegas reduces the window size by half, similar to congestion avoidance (TCP Vegas actually makes seven modifications in total over TCP Reno). Initial research⁵ indicated TCP Vegas achieves 40-70% better throughput with 1/5th to 1/2 the losses of TCP

Reno. More recent research^{6,7} indicates that performance gains are less so. While TCP Vegas is available in Linux (2.6.6 onward), SunOS (4.1.3 onward) and NetBSD (1.0 onward), it is not widely utilized because fairness (a key TCP principle) between TCP flows is not well preserved in an environment with mixed TCP Reno and TCP Vegas flows⁸. The same research indicates that TCP Vegas suffers severe performance degradation with tail-drop routers, with some improvement in routers using RED (random early discard).

TCP Window Scaling – As mentioned above, the capacity of a TCP connection is the window size divided by the round-trip time (RTT). Sometimes this is expressed as the bandwidth-delay product. As gigabit (and eventually 10 gigabit) Ethernet links start to become more common for Internet access points, TCP must scale up to these rates as well, since it is reasonable that a single TCP connection will use the entire bandwidth of the access point. If the delay through the network is constant, the only way to boost the effective capacity of the network is to increase the window size. However, TCP's window size ranges between 2 and 65,535 bytes. The TCP window scale option increases the maximum window size to 1 gigabyte. This option is indicated through a single bit in TCP header during TCP's connection-establishment phase. TCP window scaling is implemented in Windows Vista and Linux (from 2.6.8 onwards). TCP window scaling is defined in IETF RFC 1323⁹. Research¹⁰ suggests that using TCP window scaling results in bursty traffic, ultimately requiring very large buffers (thousands of packets) or traffic shaping (token bucket filters). At a more local level, many of us will discover as Windows Vista becomes common place, that smaller home routers may not support window scaling, which is turned on by default in Vista.

Compound TCP – Compound TCP is a Microsoft invention that is included with Windows Vista. Seeking to balance TCP between packet loss and packet delay, compound TCP attempts to provide good bandwidth scalability and improved RTT and TCP fairness, irrespective of window size. Compound TCP builds on TCP Reno by modifying its congestion avoidance phase to introduce a delay-based component. To complement TCP Reno's congestion window (*cwnd*), compound TCP introduces a delay window, or *dwnd*. The TCP sending window becomes the minimum of *cwnd* + *dwnd* and *awnd* (the window size advertised by the TCP receiver). Compound TCP has been documented as an IETF draft¹¹ from July 2007.

Other TCPs – The above summarizes the most dominant implementations of TCP to date. TCP remains an active research area with experiments and analysis being performed on numerous incarnations of the protocol. Some of the

experimental variants include BIC-TCP, CUBIC, FAST TCP, High Speed TCP, High Speed TCP-Low Priority, Scalable TCP, TCP Africa, TCP Hybla, TCP-Illinois, TCP-LP, TCP New Vegas, TCP VenO, Westwood, Westwood+, XCP YeAH-TCP. TCP "research" has even been the subject of an April Fools' day prank with XCP, purportedly an XML-based version of TCP. While these names are not lacking for creativity and humor, it remains to be seen whether any of them will become commonly utilized in real networks.

Queuing Management Algorithms

In this section we outline the most common queue management algorithms. Queue management is obviously about managing queues in forwarding devices such as routers and switches. Queue management should not be confused with various scheduling mechanisms such as round-robin (RR), weighted round-robin (WRR), or weighted fair queuing (WFQ), some of which quite confusingly have the word "queuing" in their names. Scheduling addresses the sequence and timing of packet transmission, and is a separate topic from queue management.

Passive Queue Management

Tail Drop – Tail drop (also sometimes known as "droptail") is perhaps the simplest queue management algorithm, where, when a queue becomes full, packets are simply dropped. Since packets can be dropped on all TCP connections simultaneously, many TCP connections on the link can be forced to go into slow-start mode. As multiple TCP connections ramp up exponentially together, congestion occurs once again, causing packets to be dropped simultaneously on many connections. This pulsating effect of TCP is called "TCP Global Synchronization". While tail drop is a very simple queue management algorithm, it historically been the most widely deployed. Tail drop can be classified as a passive queue management algorithm since it is basically a simple FIFO mechanism where packets are thrown away when queue lengths exceed buffer lengths.

Active Queue Management

The next category of schemes applies some level of intelligence to deal with queues when congestion is detected, and as such are classified as active queue management algorithms.

RED – Random Early Detection¹² was designed with four objectives in mind: (1) minimize packet loss and delay, (2) avoid global synchronization of TCP sources, (3) maintain high link utilization, and (4) remove bias against bursty sources. RED effectively signals TCP's congestion control algorithm by dropping packets before congestion occurs, causing TCP to drop its transmission rate. RED determines

what to do with packets based on the average queue length. Figure 3 illustrates the basic RED process. If the average is below some minimum threshold, packets are queued up. If the average is above some maximum threshold, packets are discarded. If the average is between the thresholds, further calculation is performed to calculate the probability that a packet will be discarded, and further thresholding is applied to either drop or mark the packet.

While RED is a huge improvement over tail drop and achieves the four objectives listed above, TCP Reno, New Reno and SACK are all known to oscillate wildly in a RED environment¹³. Much of the research around improving on RED aims at improving queue stability.

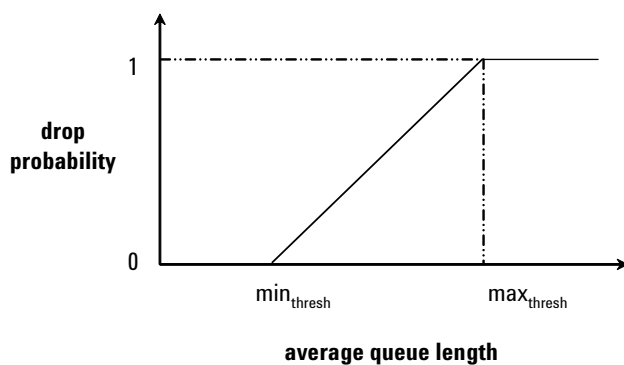


Figure 3: Drop probabilities for RED queue management

WRED – Weighted Random Early Detection is an extension of RED where the probability that packets will be dropped, is adjusted according to IP precedence levels. Typically, packets are sorted into queues based on fields such as IP precedence, either diffserv code-points or TOS values. Allowing queues to have different parameters is a simple way to implement QoS policy based on classes of traffic. Visually, we can picture WRED as supporting multiple thresholds based on weightings, as shown in Figure 4 below.

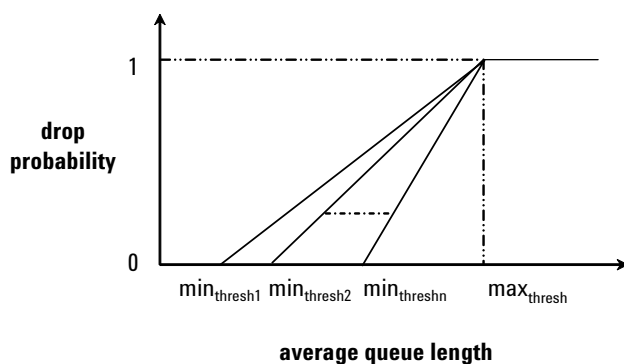


Figure 4: Drop probabilities versus WRED queue management

While both RED and WRED improve on tail drop, queue length is a poor indicator of the severity of congestion; a busy single TCP source can cause a queue to fill up as quickly as a number of less busy sources – WRED and RED cannot reliably distinguish between these two cases. WRED and RED do deal with bursty traffic to some degree by working with the average queue length, effective operation requires sufficient buffer space and judicious choice of parameters.

Other queue management schemes

While RED is by far the most commonly deployed queue management algorithm, innovation on this topic is an active research area. Here are brief descriptions of some other queue management schemes:

DRED¹⁴ – Depending strongly on the number of active TCP connections, RED can lead to queue overflows, oscillatory behavior in TCP sources and poor link utilization¹⁵. Dynamic RED seeks to stabilize TCP using a control-theoretic approach.

FRED¹⁶ – FRED or Fair Random Early Detection attempts to deal with a fundamental aspect of RED in that it imposes the same loss rate on all flows, regardless of their bandwidths. FRED also uses per-flow active accounting, and tracks the state of active flows.

SRED¹⁷ – Stabilized RED looks at sequential packets in an attempt to estimate the number of connections, and also identify potential “misbehaving flows”.

Self-Configuring RED¹⁸ -Since RED’s performance depends significantly on well-chosen parameters, researchers have proposed this mechanism to choose those parameters automatically based on a mechanism that is aware of the number of active connections and traffic load.

Blue¹⁹ – Rather than manage queues by their length, blue looks at the traffic rate (link utilization) to weight its decision to drop packets.

Pro-active Queue Management

Pro-active queue management algorithms are novel attempts to prevent (as opposed to avoid) congestion from ever happening in the first place.

GREEN²⁰ – Generalized Random Early Evasive Network (GREEN), is based on a mathematical model of steady-state TCP. GREEN is targeted at multimedia applications such as streamed or interactive audio or video which demand fixed low latency and high throughput fairness.

Yellow²¹ – This queue management scheme seeks to combine the best of RED and Blue, managing queues using both link utilization and queue length.

Other queue management schemes – The above introduces the most common queue management algorithms (tail drop, RED, WRED), as well as some quick snapshots of experimental models. Other experimental schemes include ARED, Pi, REM, VRC, AVQ, all of which attempt to overcome the shortcomings of tail drop and RED. As with TCP, the names and ideas are colorful and innovative attempts to solve real problems in managing queues.

Congestion Notification Schemes

While Explicit Congestion Notification (ECN) is neither a queue management algorithm, nor a modification to TCP, ECN is an important development in the way routers and hosts talk to each other. Rather than TCP having to infer congestion based on packet loss or time-out events, a method of explicit congestion notification or ECN has been put forward for routers to tell TCP directly that a network is in a congested state. ECN is defined in RFC 3168²², and is implemented as two bits in the diffserv IP header. While ECN is available on routers (e.g. Cisco routers since 12.2(8)T) today, it is not clear that ECN is widely turned on or not.

Complicating Factors

UDP – This paper has concentrated on the interaction between TCP and queue management algorithms. The basic dynamic is that congested routers signal to TCP to slow down, but in a fair way across all flows within a queue, and in a way that maintains good link utilization, without causing global synchronization or congestion collapse. TCP flows and UDP flows share the same networks, the same routers, the same interfaces and possibly even the same queues. While TCP and queues work together to achieve these multiple goals, UDP sources ignore congestion conditions and keep sending packets regardless of congestion levels. Many researchers have concluded that the interaction between TCP and UDP will cause UDP traffic to impact TCP traffic in an adverse manner²³.

Besides the effect of UDP on TCP, we also need to consider the opposite, i.e. the effect of TCP on UDP. Given that UDP carries the bulk of revenue-generating services such as VoIP and IPTV, TCP's detrimental effects on UDP may be the more important interaction to consider. Furthermore, as IPTV and VoIP services proliferate on the Internet causing the amount and proportion of UDP traffic to increase, it's unclear what effect this change in balance will have. While UDP appears to win out in bandwidth fairness in an environment

of mixed UDP and TCP flows, some environments appear to increase UDP delay jitter²⁴, which could critically impact delay sensitive applications such as VoIP or IPTV. At least one proposal²⁵ suggests separating TCP and UDP flows into different classes of service to solve the problem of poor fairness. According to other research²⁶, the three most common TCP variants (Reno, New Reno, SACK) will in fact cause higher loss rates (in a RED environment than in a tail-drop environment) for UDP-based applications which generate smooth traffic, ultimately requiring large play-out buffers and negating the low-delay benefit RED is meant to bring in the first place¹³.

Mischievous Applications - In spite of efforts to modify TCP or queue management to improve fairness, achieve better link utilization, and so on, an important consideration is that applications themselves are evolving to exploit the nature of networks and take an unfair share of bandwidth. For example, the open-source browser Firefox opens multiple TCP connections in attempt to manipulate the network. More widespread and problematic are peer-to-peer applications such as BitTorrent that make multiple small requests over different TCP connections, ultimately defeating the principle of fairness that TCP and queue management researchers seek to uphold. Properly managing such mischievous applications requires going beyond dealing with individual flows or connections.

Other Aspects of TCP – This white paper has focused on the congestion control part of TCP and its interaction with queue management algorithms. TCP is of course a connection-oriented protocol, and congestion can naturally affect the connection establishment or teardown phases of a connection. TCP also contains timing aspects such as Karn's algorithm which is used to adjust the round-trip time-out throughout the life of a connection. In multiple successive attempts to retransmit a lost packet or re-establish a connection, TCP's interval between attempts grows exponentially. While these aspects of TCP are beyond the scope of this paper, congestion and packet loss will interact with TCP's connectivity and timing issues, and ultimately the overall performance of the network. In other words, TCP congestion management and queue management algorithms are not the last word on performance.

Diverse Traffic – Besides various TCP mechanisms, the nature of TCP traffic itself has an effect on performance. It is clear that the number of active TCP connections is an important parameter²⁷. The duration of TCP flows (sometimes characterized as "The War between Mice and Elephants"²⁸) is likewise another important consideration in that short-lived flows (the mice) tend to receive less than their fair share of bandwidth vis-à-vis long-lived flows (the elephants).

Wrapping Up and Looking Ahead

We've covered the basics of TCP congestion control, including the most widely deployed variants (Reno), as well as some soon-to-be deployed (compound TCP) or experimental variants (many). Improvements to TCP attempt to meet the goals of high link utilization, robustness, and responsiveness while avoiding large problems like congestion collapse, or chronic congestion. TCP is also evolving to take advantage of new opportunities such as higher bandwidth access networks (via window scaling) and high-delay environments such as wireless (via compound TCP).

Likewise, we've looked at the most common queue management algorithms (tail drop, RED), as well as several experimental variants (many) that attempt to improve on shortcomings, such as fairness, parameter setting, susceptibility to the number of active TCP connections, responsiveness for multimedia, as well as the jitter impact on UDP traffic in the same queue or link. We've also touched on the closely related topics of ECN and the thorny problem of applications which deliberately thwart TCP's effectiveness and attempts by the network to offer fairness.

Besides what's published as academic research or as an RFC from the IETF, we should also look to commercial interests to provide innovative solutions to solve, or at least manage some of these challenges. Looking ahead, it seems hard to avoid considering intelligent options provided by deep-packet inspection solutions²⁹, as well as next-generation routers³⁰ which track the state of every single TCP flow.

The amount of interest from both academic and commercial interests in TCP and queue management is a strong indicator that problems in both of these intimately related areas are far from being solved. With the goal of closing the gap between networks which perform well in the evaluation lab and networks which perform well under realistic traffic, we should expect continued innovation from all angles.

References

1. Congestion Avoidance and Control, V. Jacobson, M. Karels, SIGCOMM, November 1988.
2. TCP Congestion Control, M. Allman, V. Paxson, W. Stevens, IETF RFC 2581, April 1999.
3. The NewReno Modification to TCP's Fast Recovery Algorithm, S. Floyd, T. Henderson, A. Gurtov, IETF RFC 3782, April 2004.
4. TCP Selective Acknowledgment Options, M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, IETF RFC 2018, October 1996.
5. TCP Vegas: New Techniques for Congestion Detection and Avoidance – L. Brakmo, S. O'Malley and L. Peterson, Proceedings of the SIGCOMM '94 Symposium, August 1994, pg. 24-35
6. TCP Vegas Revisited – U. Hengartner, J. Bolliger, Th. Gross, CS598RHK – Spring 2006.
7. TCP New Vegas: Performance Evaluation and Validation, 11th IEEE Symposium on Computers and Communications (ISCC '06), pp. 541-546.
8. Fairness Comparisons between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas – K. Kurata, G. Hasegawa, M. Murata; INET 2000, July 2000.
9. TCP Extensions for High Performance, V. Jacobson, R. Braden, D. Borman, IETF RFC 1323, May 1992.
10. Performance Measure and Analysis of Large TCP window effect in broadband network including GEO satellite network, Dong Joon Choi, Nae Soo Kim (ETRI), HSNMC '02, 5th International Conference on High-Speed Networks and Multimedia Communications, July 2002.
11. Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks (draft –sridharan-tcpm-ctcp-00.txt), M. Sridharan, K. Tan, D. Bansal, D. Thaler; IETF RFC.
12. Random Early Detection gateways for Congestion Avoidance, Floyd, S., and Jacobson, V., V.1 N.4, August 1993, pp. 397-413.
13. Analytic evaluation of RED performance – M. May, T. Bonald, J. Bolot; Proceedings of IEEE Infocom 2000.
14. A control theoretic approach to active queue management, J. Aweya, M. Oulette, D. Montuno, Computer Networks, Volume 36, Issues 2-3, pp. 203-225, July 2001.
15. Dynamics of Random Early Detection, D. Lin, R. Morris, SIGCOMM '97, 1997.
16. FRED – fair random early detection algorithm for TCP over ATM networks, W. Kim, B. Lee, Electronics Letters, Vol. 34, January 1988.
17. SRED: Stabilized RED, T. Ott, T. Lakshman, L. Wong, Proceedings of IEEE INFOCOM, pp. 1346-1355, March 1999.
18. A Self-Configuring RED Gateway for Quality of Service (QoS) Networks, C. Chien, W. Liao, IEEE/ICME 2003.
19. The Blue active queue management algorithms, W. Feng, K. Shin, D. Kandlur, D. Saha, IEEE/ACM Transactions on Networking 10, pp. 513-528, 2002.
20. GREEN: Proactive Queue Management over a Best-Effort Network, W. Feng, A. Kapadia, S. Thulasidasan, Globeco02, 2002.
21. The Yellow active queue management algorithm, C. Long, B. Zhao, X. Guan, J. Yang, Computer Networks, Volume 47, Issue 4, March 2005.
22. The Addition of Explicit Congestion Notification (ECN) to IP, K. Ramakrishnan, S. Floyd, RFC 3188, January 1999.
23. The Impacts of Aggregation on the Performance of TCP Flows in DS Networks, Laatu, V., Harju, J., Loula, P., ICEFIN Research Laboratory, ICN 2004.
24. TCP and UDP Performance for Internet over Optical Packet-Switched Networks, He, J., Chan, S., Proceedings of the IEEE ICC, Volume 2, pages 1350-1354, 2003.
25. Study of the TCP/UDP fairness issue for the assured forwarding per hop behavior in differentiated services networks, A. Kolarov, 2001 IEEE Workshop on High Performance Switching and Routing, 2001.
26. Performance Comparison of TCP Implementations in QoS Provisioning Networks – H. Koga, Yoshiaki Hori, Yuji Oie; INET 2000, July 2000.
27. Stability analysis of RED gateway with multiple TCP Reno connections, X. Chen, S. Wong, C. Tse and L. Trajkovic, Proc. IEEE Int. Symp. Circuits and Systems, New Orleans, LA, May 2007, pp. 1429-1432.
28. The War Between Mice and Elephants, L. Guo, I. Matta, Technical Report BU-CS-2001-005, Boston University, Computer Science Department, May 2001.
29. Insider: P2P Drives Use of DPI, Darkreading article, September 2006.
30. Intelligent Flow Routing for Economical Delivery of Next-Generation Services, L. Roberts, Anagran White Paper, 2007.

This page intentionally left blank.

Sales, Service and Support

United States:

Agilent Technologies
Test and Measurement Call Center
P.O. Box 4026
Englewood, CO 80155-4026
1-800-452-4844

Canada:

Agilent Technologies Canada Inc.
2660 Matheson Blvd. E
Mississauga, Ontario
L4V 5M2
1-877-894-4414

Europe:

Agilent Technologies
European Marketing Organisation
P.O. Box 999
1180 AZ Amstelveen
The Netherlands
(31 20) 547-2323

United Kingdom

07004 666666

Japan:

Agilent Technologies Japan Ltd.
Measurement Assistance Center
9-1, Takakura-Cho, Hachioji-Shi,
Tokyo 192-8510, Japan
Tel: (81) 426-56-7832
Fax: (81) 426-56-7840

Latin America:

Agilent Technologies
Latin American Region Headquarters
5200 Blue Lagoon Drive, Suite #950
Miami, Florida 33126
U.S.A.
Tel: (305) 269-7500
Fax: (305) 267-4286

Asia Pacific:

Agilent Technologies
19/F, Cityplaza One, 1111 King's Road,
Taikoo Shing, Hong Kong, SAR
Tel: (852) 3197-7777
Fax: (852) 2506-9233

Australia/New Zealand:

Agilent Technologies Australia Pty Ltd
347 Burwood Highway
Forest Hill, Victoria 3131
Tel: 1-800-629-485 (Australia)
Fax: (61-3) 9272-0749
Tel: 0-800-738-378 (New Zealand)
Fax: (64-4) 802-6881

This information is subject to change without notice.

Printed on recycled paper

© Agilent Technologies, Inc. 2008

Printed in USA January 21, 2008

5989-7873EN

