Network Working Group
Request for Comments: 3150
BCP: 48
Category: Best Current Practice

S. Dawkins
G. Montenegro
M . Kojo
V. Magret

July 2001

End-to-end Performance Implications of Slow Links

Status of this Memo

Copyright Notice

Abstract

This document makes performance-related recommendations for users of network paths that traverse "very low bit-rate" links.

"Very low bit-rate" implies "slower than we would like". This recommendation may be useful in any network where hosts can saturate available bandwidth, but the design space for this recommendation explicitly includes connections that traverse 56 Kb/second modem links or 4.8 Kb/second wireless access links - both of which are widely deployed.

This document discusses general-purpose mechanisms. Where application-specific mechanisms can outperform the relevant general-purpose mechanism, we point this out and explain why.

This document has some recommendations in common with RFC 2689, "Providing integrated services over low-bitrate links", especially in areas like header compression. This document focuses more on traditional data applications for which "best-effort delivery" is appropriate.

Table of Contents

1.0 Introduction

The Internet protocol stack was designed to operate in a wide range of link speeds, and has met this design goal with only a limited number of enhancements (for example, the use of TCP window scaling as described in "TCP Extensions for High Performance" [RFC1323] for very-high-bandwidth connections).

Pre-World Wide Web application protocols tended to be either interactive applications sending very little data (e.g., Telnet) or bulk transfer applications that did not require interactive response (e.g., File Transfer Protocol, Network News). The World Wide Web has given us traffic that is both interactive and often "bulky", including images, sound, and video.

The World Wide Web has also popularized the Internet, so that there is significant interest in accessing the Internet over link speeds that are much "slower" than typical office network speeds. In fact, a significant proportion of the current Internet users is connected to the Internet over a relatively slow last-hop link. In future, the number of such users is likely to increase rapidly as various mobile devices are foreseen to to be attached to the Internet over slow wireless links.

In order to provide the best interactive response for these "bulky" transfers, implementors may wish to minimize the number of bits actually transmitted over these "slow" connections. There are two

areas that can be considered - compressing the bits that make up the overhead associated with the connection, and compressing the bits that make up the payload being transported over the connection.

In addition, implementors may wish to consider TCP receive window settings and queuing mechanisms as techniques to improve performance over low-speed links. While these techniques do not involve protocol changes, they are included in this document for completeness.

2.0 Description of Optimizations

This section describes optimizations which have been suggested for use in situations where hosts can saturate their links. The next section summarizes recommendations about the use of these optimizations.

2.1 Header Compression Alternatives

Mechanisms for TCP and IP header compression defined in [RFC1144, RFC2507, RFC2508, RFC2509, RFC3095] provide the following benefits:

- Improve interactive response time
- Decrease header overhead (for a typical dialup MTU of 296 bytes, the overhead of TCP/IP headers can decrease from about 13 percent with typical 40-byte headers to 1-1.5 percent with with 3-5 byte compressed headers, for most packets). This enables use of small packets for delay-sensitive low data-rate traffic and good line efficiency for bulk data even with small segment sizes (for reasons to use a small MTU on slow links, see section 2.3)
- Many slow links today are wireless and tend to be significantly lossy. Header compression reduces packet loss rate over lossy links (simply because shorter transmission times expose packets to fewer events that cause loss).

[RFC1144] header compression is a Proposed Standard for TCP Header compression that is widely deployed. Unfortunately it is vulnerable on lossy links, because even a single bit error results in loss of synchronization between the compressor and decompressor. It uses TCP timeouts to detect a loss of such synchronization, but these errors result in loss of data (up to a full TCP window), delay of a full RTO, and unnecessary slow-start.

A more recent header compression proposal [RFC2507] includes an explicit request for retransmission of an uncompressed packet to allow resynchronization without waiting for a TCP timeout (and executing congestion avoidance procedures). This works much better on links with lossy characteristics.

The above scheme ceases to perform well under conditions as extreme as those of many cellular links (error conditions of 1e-3 or 1e-2 and round trip times over 100 ms.). For these cases, the 'Robust Header Compression' working group has developed ROHC [RFC3095]. Extensions of ROHC to support compression of TCP headers are also under development.

[RFC1323] defines a "TCP Timestamp" option, used to prevent "wrapping" of the TCP sequence number space on high-speed links, and to improve TCP RTT estimates by providing unambiguous TCP roundtrip timings. Use of TCP timestamps prevents header compression, because the timestamps are sent as TCP options. This means that each timestamped header has TCP options that differ from the previous header, and headers with changed TCP options are always sent uncompressed. In addition, timestamps do not seem to have much of an impact on RTO estimation [AlPa99].

Nevertheless, the ROHC working group is developing schemes to compress TCP headers, including options such as timestamps and selective acknowledgements.

Recommendation: Implement [RFC2507], in particular as it relates to IPv4 tunnels and Minimal Encapsulation for Mobile IP, as well as TCP header compression for lossy links and links that reorder packets. PPP capable devices should implement "IP Header Compression over PPP" [RFC2509]. Robust Header Compression [RFC3095] is recommended for extremely slow links with very high error rates (see above), but implementors should judge if its complexity is justified (perhaps by the cost of the radio frequency resources).

[RFC1144] header compression should only be enabled when operating over reliable "slow" links.

Use of TCP Timestamps [RFC1323] is not recommended with these connections, because it complicates header compression. Even though the Robust Header Compression (ROHC) working group is developing specifications to remedy this, those mechanisms are not yet fully developed nor deployed, and may not be generally justifiable. Furthermore, connections traversing "slow" links do not require protection against TCP sequence-number wrapping.

2.2 Payload Compression Alternatives

Compression of IP payloads is also desirable on "slow" network links. "IP Payload Compression Protocol (IPComp)" [RFC2393] defines a framework where common compression algorithms can be applied to arbitrary IP segment payloads.

IP payload compression is something of a niche optimization. It is necessary because IP-level security converts IP payloads to random bitstreams, defeating commonly-deployed link-layer compression mechanisms which are faced with payloads that have no redundant "information" that can be more compactly represented.

However, many IP payloads are already compressed (images, audio, video, "zipped" files being transferred), or are already encrypted above the IP layer (e.g., SSL [SSL]/TLS [RFC2246]). These payloads will not "compress" further, limiting the benefit of this optimization.

For uncompressed HTTP payload types, HTTP/1.1 [RFC2616] also includes Content-Encoding and Accept-Encoding headers, supporting a variety of compression algorithms for common compressible MIME types like text/plain. This leaves only the HTTP headers themselves uncompressed.

In general, application-level compression can often outperform IPComp, because of the opportunity to use compression dictionaries based on knowledge of the specific data being compressed.

Extensive use of application-level compression techniques will reduce the need for IPComp, especially for WWW users.

Recommendation: IPComp may optionally be implemented.

2.3 Choosing MTU Sizes

There are several points to keep in mind when choosing an MTU for low-speed links.

First, if a full-length MTU occupies a link for longer than the delayed ACK timeout (typically 200 milliseconds, but may be up to 500 milliseconds), this timeout will cause an ACK to be generated for every segment, rather than every second segment, as occurs with most implementations of the TCP delayed ACK algorithm.

Second, "relatively large" MTUs, which take human-perceptible amounts of time to be transmitted into the network, create human-perceptible delays in other flows using the same link. [RFC1144] considers 100-200 millisecond delays as human-perceptible. The convention of choosing 296-byte MTUs (with header compression enabled) for dialup access is a compromise that limits the maximum link occupancy delay with full-length MTUs close to 200 milliseconds on 9.6 Kb/second links.

Third, on last-hop links using a larger link MTU size, and therefore larger MSS, would allow a TCP sender to increase its congestion window faster in bytes than when using a smaller MTU size (and a smaller MSS). However, with a smaller MTU size, and a smaller MSS size, the congestion window, when measured in segments, increases more quickly than it would with a larger MSS size. Connections using smaller MSS sizes are more likely to be able to send enough segments to generate three duplicate acknowledgements, triggering fast retransmit/fast recovery when packet losses are encountered. Hence, a smaller MTU size is useful for slow links with lossy characteristics.

Fourth, using a smaller MTU size also decreases the queuing delay of a TCP flow (and thereby RTT) compared to use of larger MTU size with the same number of packets in a queue. This means that a TCP flow using a smaller segment size and traversing a slow link is able to inflate the congestion window (in number of segments) to a larger value while experiencing the same queuing delay.

Finally, some networks charge for traffic on a per-packet basis, not on a per-kilobyte basis. In these cases, connections using a larger MTU may be charged less than connections transferring the same number of bytes using a smaller MTU.

Recommendation: If it is possible to do so, MTUs should be chosen that do not monopolize network interfaces for human-perceptible amounts of time, and implementors should not chose MTUs that will occupy a network interface for significantly more than 100-200 milliseconds.

2.4 Interactions with TCP Congestion Control [RFC2581]

In many cases, TCP connections that traverse slow links have the slow link as an "access" link, with higher-speed links in use for most of the connection path. One common configuration might be a laptop computer using dialup access to a terminal server (a last-hop router), with an HTTP server on a high-speed LAN "behind" the terminal server.

In this case, the HTTP server may be able to place packets on its directly-attached high-speed LAN at a higher rate than the last-hop router can forward them on the low-speed link. When the last-hop router falls behind, it will be unable to buffer the traffic intended for the low-speed link, and will become a point of congestion and begin to drop the excess packets. In particular, several packets may be dropped in a single transmission window when initial slow start overshoots the last-hop router buffer.

Although packet loss is occurring, it isn't detected at the TCP sender until one RTT time after the router buffer space is exhausted and the first packet is dropped. This late congestion signal allows the congestion window to increase up to double the size it was at the time the first packet was dropped at the router.

If the link MTU is large enough to take more than the delayed ACK timeout interval to transmit a packet, an ACK is sent for every segment and the congestion window is doubled in a single RTT. If a smaller link MTU is in use and delayed ACKs can be utilized, the congestion window increases by a factor of 1.5 in one RTT. In both cases the sender continues transmitting packets well beyond the congestion point of the last-hop router, resulting in multiple packet losses in a single window.

The self-clocking nature of TCP's slow start and congestion avoidance algorithms prevent this buffer overrun from continuing. In addition, these algorithms allow senders to "probe" for available bandwidth - cycling through an increasing rate of transmission until loss occurs, followed by a dramatic (50-percent) drop in transmission rate. This happens when a host directly connected to a low-speed link offers an advertised window that is unrealistically large for the low-speed link. During the congestion avoidance phase the peer host continues to probe for available bandwidth, trying to fill the advertised window, until packet loss occurs.

The same problems may also exist when a sending host is directly connected to a slow link as most slow links have some local buffer in the link interface. This link interface buffer is subject to overflow exactly in the same way as the last-hop router buffer.

When a last-hop router with a small number of buffers per outbound link is used, the first buffer overflow occurs earlier than it would if the router had a larger number of buffers. Subsequently with a smaller number of buffers the periodic packet losses occur more frequently during congestion avoidance, when the sender probes for available bandwidth.

The most important responsibility of router buffers is to absorb bursts. Too few buffers (for example, only three buffers per outbound link as described in [RFC2416]) means that routers will overflow their buffer pools very easily and are unlikely to absorb even a very small burst. When a larger number of router buffers are allocated per outbound link, the buffer space does not overflow as quickly but the buffers are still likely to become full due to TCP's default behavior. A larger number of router buffers leads to longer queuing delays and a longer RTT.

If router queues become full before congestion is signaled or remain full for long periods of time, this is likely to result in "lock-out", where a single connection or a few connections occupy the router queue space, preventing other connections from using the link [RFC2309], especially when a tail drop queue management discipline is being used.

Therefore, it is essential to have a large enough number of buffers in routers to be able to absorb data bursts, but keep the queues normally small. In order to achieve this it has been recommended in [RFC2309] that an active queue management mechanism, like Random Early Detection (RED) [RED93], should be implemented in all Internet routers, including the last-hop routers in front of a slow link. It should also be noted that RED requires a sufficiently large number of router buffers to work properly. In addition, the appropriate parameters of RED on a last-hop router connected to a slow link will likely deviate from the defaults recommended.

Active queue management mechanism do not eliminate packet drops but, instead, drop packets at earlier stage to solve the full-queue problem for flows that are responsive to packet drops as congestion signal. Hosts that are directly connected to low-speed links may limit the receive windows they advertise in order to lower or eliminate the number of packet drops in a last-hop router. When doing so one should, however, take care that the advertised window is large enough to allow full utilization of the last-hop link capacity and to allow triggering fast retransmit, when a packet loss is encountered. This recommendation takes two forms:

- Modern operating systems use relatively large default TCP receive buffers compared to what is required to fully utilize the link capacity of low-speed links. Users should be able to choose the default receive window size in use - typically a system-wide parameter. (This "choice" may be as simple as "dial-up access/LAN access" on a dialog box - this would accommodate many environments without requiring hand-tuning by experienced network engineers.)

- Application developers should not attempt to manually manage network bandwidth using socket buffer sizes. Only in very rare circumstances will an application actually know both the bandwidth and delay of a path and be able to choose a suitably low (or high) value for the socket buffer size to obtain good network performance.

This recommendation is not a general solution for any network path that might involve a slow link. Instead, this recommendation is applicable in environments where the host "knows" it is always connected to other hosts via "slow links". For hosts that may connect to other host over a variety of links (e.g., dial-up laptop computers with LAN-connected docking stations), buffer auto-tuning for the receive buffer is a more reasonable recommendation, and is discussed below.

2.5 TCP Buffer Auto-tuning

[SMM98] recognizes a tension between the desire to allocate "large" TCP buffers, so that network paths are fully utilized, and a desire to limit the amount of memory dedicated to TCP buffers, in order to efficiently support large numbers of connections to hosts over network paths that may vary by six orders of magnitude.

The technique proposed is to dynamically allocate TCP buffers, based on the current congestion window, rather than attempting to preallocate TCP buffers without any knowledge of the network path.

This proposal results in receive buffers that are appropriate for the window sizes in use, and send buffers large enough to contain two windows of segments, so that SACK and fast recovery can recover losses without forcing the connection to use lengthy retransmission timeouts.

While most of the motivation for this proposal is given from a server's perspective, hosts that connect using multiple interfaces with markedly-different link speeds may also find this kind of technique useful. This is true in particular with slow links, which are likely to dominate the end-to-end RTT. If the host is connected only via a single slow link interface at a time, it is fairly easy to (dynamically) adjust the receive window (and thus the advertised window) to a value appropriate for the slow last-hop link with known bandwidth and delay characteristics.

Recommendation: If a host is sometimes connected via a slow link but the host is also connected using other interfaces with markedly-different link speeds, it may use receive buffer auto-tuning to adjust the advertised window to an appropriate value.

2.6 Small Window Effects

If a TCP connection stabilizes with a congestion window of only a few segments (as could be expected on a "slow" link), the sender isn't sending enough segments to generate three duplicate acknowledgements, triggering fast retransmit and fast recovery. This means that a retransmission timeout is required to repair the loss - dropping the TCP connection to a congestion window with only one segment.

[TCPB98] and [TCPF98] observe that (in studies of network trace datasets) it is relatively common for TCP retransmission timeouts to occur even when some duplicate acknowledgements are being sent. The challenge is to use these duplicate acknowledgements to trigger fast retransmit/fast recovery without injecting traffic into the network unnecessarily - and especially not injecting traffic in ways that will result in instability.

The "Limited Transmit" algorithm [RFC3042] suggests sending a new segment when the first and second duplicate acknowledgements are received, so that the receiver is more likely to be able to continue to generate duplicate acknowledgements until the TCP retransmit threshold is reached, triggering fast retransmit and fast recovery. When the congestion window is small, this is very useful in assisting fast retransmit and fast recovery to recover from a packet loss without using a retransmission timeout. We note that a maximum of two additional new segments will be sent before the receiver sends either a new acknowledgement advancing the window or two additional duplicate acknowledgements, triggering fast retransmit/fast recovery, and that these new segments will be acknowledgement-clocked, not back-to-back.

Recommendation: Limited Transmit should be implemented in all hosts.

3.0 Summary of Recommended Optimizations

This section summarizes our recommendations regarding the previous standards-track mechanisms, for end nodes that are connected via a slow link.

Header compression should be implemented. [RFC1144] header compression can be enabled over robust network links. [RFC2507] should be used over network connections that are expected to experience loss due to corruption as well as loss due to congestion. For extremely lossy and slow links, implementors should evaluate ROHC [RFC3095] as a potential solution. [RFC1323] TCP timestamps must be turned off because (1) their protection against TCP sequence number wrapping is unjustified for slow links, and (2) they complicate TCP header compression.

IP Payload Compression [RFC2393] should be implemented, although compression at higher layers of the protocol stack (for example [RFC 2616]) may make this mechanism less useful.

For HTTP/1.1 environments, [RFC2616] payload compression should be implemented and should be used for payloads that are not already compressed.

Implementors should choose MTUs that don't monopolize network interfaces for more than 100-200 milliseconds, in order to limit the impact of a single connection on all other connections sharing the network interface.

Use of active queue management is recommended on last-hop routers that provide Internet access to host behind a slow link. In addition, number of router buffers per slow link should be large enough to absorb concurrent data bursts from more than a single flow. To absorb concurrent data bursts from two or three TCP senders with a typical data burst of three back-to-back segments per sender, at least six (6) or nine (9) buffers are needed. Effective use of active queue management is likely to require even larger number of buffers.

Implementors should consider the possibility that a host will be directly connected to a low-speed link when choosing default TCP receive window sizes.

Application developers should not attempt to manually manage network bandwidth using socket buffer sizes as only in very rare circumstances an application will be able to choose a suitable value for the socket buffer size to obtain good network performance.

Limited Transmit [RFC3042] should be implemented in all end hosts as it assists in triggering fast retransmit when congestion window is small.

All of the mechanisms described above are stable standards-track RFCs (at Proposed Standard status, as of this writing).

In addition, implementors may wish to consider TCP buffer auto-tuning, especially when the host system is likely to be used with a wide variety of access link speeds. This is not a standards-track TCP mechanism but, as it is an operating system implementation issue, it does not need to be standardized.

Of the above mechanisms, only Header Compression (for IP and TCP) may cease to work in the presence of end-to-end IPSEC. However, [RFC3095] does allow compressing the ESP header.

4.0 Topics For Further Work

In addition to the standards-track mechanisms discussed above, there are still opportunities to improve performance over low-speed links.

"Sending fewer bits" is an obvious response to slow link speeds. The now-defunct HTTP-NG proposal [HTTP-NG] replaced the text-based HTTP header representation with a binary representation for compactness. However, HTTP-NG is not moving forward and HTTP/1.1 is not being enhanced to include a more compact HTTP header representation. Instead, the Wireless Application Protocol (WAP) Forum has opted for the XML-based Wireless Session Protocol [WSP], which includes a compact header encoding mechanism.

It would be nice to agree on a more compact header representation that will be used by all WWW communities, not only the wireless WAN community. Indeed, general XML content encodings have been proposed [Millau], although they are not yet widely adopted.

We note that TCP options which change from segment to segment effectively disable header compression schemes deployed today, because there's no way to indicate that some fields in the header are unchanged from the previous segment, while other fields are not. The Robust Header Compression working group is developing such schemes for TCP options such as timestamps and selective acknowledgements. Hopefully, documents subsequent to [RFC3095] will define such specifications.

Another effort worth following is that of 'Delta Encoding'. Here, clients that request a slightly modified version of some previously cached resource would receive a succinct description of the differences, rather than the entire resource [HTTP-DELTA].

5.0 Security Considerations

All recommendations included in this document are stable standards-track RFCs (at Proposed Standard status, as of this writing) or otherwise do not suggest any changes to any protocol. With the exception of Van Jacobson compression [RFC1144] and [RFC2507, RFC2508, RFC2509], all other mechanisms are applicable to TCP connections protected by end-to-end IPSec. This includes ROHC [RFC3095], albeit partially, because even though it can compress the outermost ESP header to some extent, encryption still renders any payload data uncompressible (including any subsequent protocol headers).

6.0 IANA Considerations

This document is a pointer to other, existing IETF standards. There are no new IANA considerations.

7.0 Acknowledgements

This recommendation has grown out of "Long Thin Networks" [RFC2757], which in turn benefited from work done in the IETF TCPSAT working group.

8.0 References

[AlPa99] Mark Allman and Vern Paxson, "On Estimating End-to-End Network Path Properties", in ACM SIGCOMM 99 Proceedings, 1999.

[HTTP-DELTA] J. Mogul, et al., "Delta encoding in HTTP", Work in Progress.

[HTTP-NG] Mike Spreitzer, Bill Janssen, "HTTP 'Next Generation'", 9th International WWW Conference, May, 2000. Also available as: http://www.www9.org/w9cdrom/60/60.html

[Millau] Marc Girardot, Neel Sundaresan, "Millau: an encoding format for efficient representation and exchange of XML over the Web", 9th International WWW Conference, May, 2000. Also available as: http://www.www9.org/w9cdrom/154/154.html

[PAX97] Paxson, V., "End-to-End Internet Packet Dynamics", 1997, in SIGCOMM 97 Proceedings, available as: http://www.acm.org/sigcomm/ccr/archive/ccr-toc/ccr-toc-97.html

[RED93] Floyd, S., and Jacobson, V., Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413. Also available from http://ftp.ee.lbl.gov/floyd/red.html.

[RFC1144] Jacobson, V., "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC 1144, February 1990.

[RFC1323] Jacobson, V., Braden, R. and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol: Version 1.0", RFC 2246, January 1999.

[RFC2309] Braden, R., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J. and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2393] Shacham, A., Monsour, R., Pereira, R. and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 2393, December 1998.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2416] Shepard, T. and C. Partridge, "When TCP Starts Up With Four Packets Into Only Three Buffers", RFC 2416, September 1998.

[RFC2507] Degermark, M., Nordgren, B. and S. Pink, "IP Header Compression", RFC 2507, February 1999.

[RFC2508] Casner, S. and V. Jacobson. "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, February 1999.

[RFC2509] Engan, M., Casner, S. and C. Bormann, "IP Header Compression over PPP", RFC 2509, February 1999.

[RFC2581] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, January 2000.

[RFC3042] Allman, M., Balakrishnan, H. and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

[RFC3095] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T. and H. Zheng, "RObust Header Compression (ROHC): Framework and four Profiles: RTP, UDP ESP and uncompressed", RFC 3095, July 2001.

[SMM98] Jeffrey Semke, Matthew Mathis, and Jamshid Mahdavi, "Automatic TCP Buffer Tuning", in ACM SIGCOMM 98 Proceedings 1998. Available from http://www.acm.org/sigcomm/sigcomm98/tp/abs_26.html.

[SSL] Alan O. Freier, Philip Karlton, Paul C. Kocher, The SSL Protocol: Version 3.0, March 1996. (Expired Internet-Draft, available from http://home.netscape.com/eng/ssl3/ssl-toc.html)

[TCPB98] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, Randy H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements", IEEE Infocom, March 1998. Available from: http://www.cs.berkeley.edu/~hari/papers/infocom98.ps.gz

[TCPF98] Dong Lin and H.T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements", IEEE Infocom, March 1998. Available from: http://www.eecs.harvard.edu/networking/papers/ infocom-tcp-final-198.pdf

[WSP] Wireless Application Protocol Forum, "WAP Wireless Session Protocol Specification", approved 4 May, 2000, available from http://www1.wapforum.org/tech/documents/WAP-203-WSP-20000504-a.pdf. (informative reference).

Authors' Addresses

Questions about this document may be directed to:

Spencer Dawkins
Fujitsu Network Communications
2801 Telecom Parkway
Richardson, Texas 75082

Phone: +1-972-479-3782
EMail: spencer.dawkins@fnc.fujitsu.com

Gabriel Montenegro
Sun Microsystems Laboratories, Europe
29, chemin du Vieux Chene
38240 Meylan, FRANCE

Phone: +33 476 18 80 45
EMail: gab@sun.com

Markku Kojo
Department of Computer Science
University of Helsinki
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 HELSINKI
Finland

Phone: +358-9-1914-4179
Fax: +358-9-1914-4441
EMail: kojo@cs.helsinki.fi

Vincent Magret
Alcatel Internetworking, Inc.
26801 W. Agoura road
Calabasas, CA, 91301

Phone: +1 818 878 4485
EMail: vincent.magret@alcatel.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.