TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Updates: 793 (if approved)
Intended status: Standards Track
Expires: September 12, 2019

F. Gont
UTN-FRH / SI6 Networks
D. Borman
Quantum Corporation

March 11, 2019

On the Validation of TCP Sequence Numbers
draft-gont-tcpm-tcp-seq-validation-04.txt

Abstract

When TCP receives packets that lie outside of the receive window, the corresponding packets are dropped and either an ACK, RST or no response is generated due to the out-of-window packet, with no further processing of the packet. Most of the time, this works just fine and TCP remains stable, especially when a TCP connection has unidirectional data flow. However, there are three scenarios in which packets that are outside of the receive window should still have their ACK field processed, or else a packet war will take place. The aforementioned issues have affected a number of popular TCP implementations, typically leading to connection failures, system crashes, or other undesirable behaviors. This document describes the three scenarios in which the aforementioned issues might arise, and formally updates RFC 793 such that these potential problems are mitigated.

Table of Contents

1. Introduction

TCP processes incoming packets in in-sequence order. Packets that are not in-sequence but have data that lies in the receive window are queued for later processing. Packets that lie outside of the receive window are dropped and either an ACK, RST or no response is generated due to the out-of-window packet, with no further processing of the packet. Most of the time, this works just fine and TCP remains stable, especially when a TCP connection has unidirectional data flow.

However, there are three situations in which packets that are outside of the receive window should still have their ACK field processed. These situations arise during a simultaneous open, simultaneous window probes and a simultaneous close. In all three of these cases, a packet will arrive with a sequence number that is one to the left of the window, but the acknowledgement field has updated information that needs to be processed to avoid entering a packet war, in which both sides of the connection generate a response to the received packet, which just causes the other side to do the same thing. This issue has affected a number of popular TCP implementations, typically leading to connection failures, system crashes, or other undesirable behaviors.

Section 2 provides an overview of the TCP sequence number validation checks specified in RFC 793. Section 3 describes the three scenarios in which the current TCP sequence number validation checks can lead to undesirable behaviors. Section 4 formally updates RFC 793 such that these issues are mitigated.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. TCP Sequence Number Validation

Section 3.3 of RFC 793 [RFC0793] specifies (in pp. 25-26) how the TCP sequence number of incoming segments is to be validated. It summarizes the validation of the TCP sequence number with the following table:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

RFC 793 states that if an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set), and that after sending the acknowledgment, the unacceptable segment should be dropped.

Section 3.9 of RFC 793 repeats (in pp. 69-76) the same validation checks when describing the processing of incoming TCP segments meant for connections that are in the SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, or TIME-WAIT states (i.e., any state other than CLOSED, LISTEN, or SYN-SENT).

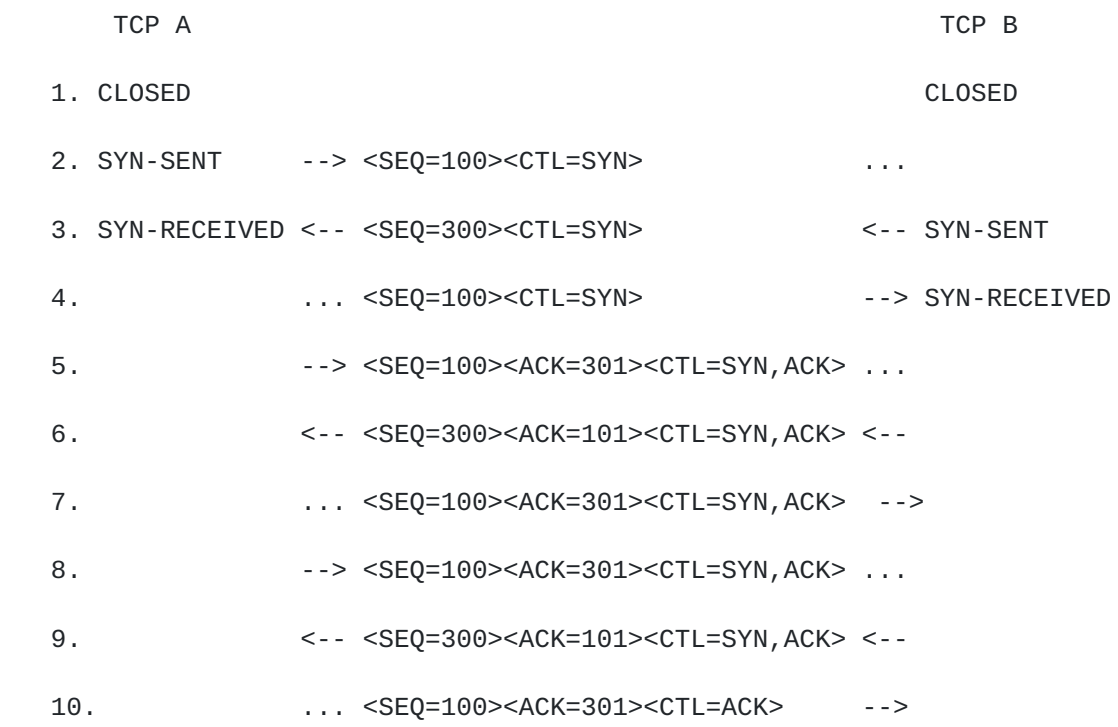
A key problem with the aforementioned checks is that it assumes that a segment must be processed only if a portion of it overlaps with the receive window. However, there are some cases in which the Acknowledgement information in an incoming segment needs to be processed by TCP even if the contents of the segment does not overlap with the receive window. Otherwise, the TCP state machine may become dead-locked, and this situation may result in undesirable behaviors such as system crashes.

3. Scenarios in which Undesirable Behaviors Might Arise

The following subsections describe the three scenarios in which the TCP Sequence Number validation specified n RFC 793 (and described in Section 2 of this document) could result in undesirable behaviors.

3.1. TCP simultaneous open

The following figure illustrates a typical "simultaneous open" attempt.



(Failed) Simultaneous Connection Synchronization

In line 2, TCP A performs an "active open" by sending a SYN segment to TCP B, and enters the SYN-SENT state. In line 3, TCP B performs an "active open" by sending a SYN segment to TCP A, and enters the "SYN-SENT" state; when TCP A receives this SYN segment sent by TCP B, it enters the SYN-RECEIVED state, and its RCV.NXT becomes 301. In line 4, similarly, when TCP B receives the SYN segment sent by TCP A, it enters the SYN-RECEIVED STATE and its RCV.NXT becomes 101. In line 5, TCP A sends a SYN/ACK in response to the received SYN segment from line 3. In line 6, TCP B sends a SYN/ACK in response to the received SYN segment from line 4. In line 7, TCP B receives the SYN/ACK from line 5. In line 8, TCP A receives the SYN/ACK from line 6, which fails the TCP Sequence Number validation check. As a result, the received packet is dropped, and a SYN/ACK is sent in response. In line 9, TCP B processes the SYN/ACK from line 7, which fails the TCP Sequence Number validation check. As a result, the received packet is dropped, and a SYN/ACK is sent in response. In line 10, the SYN/ACK from line 9 arrives at TCP B. The segment exchange from lines 8-10 will continue forever (with both TCP end-points will be stuck in the SYN-RECEIVED state), thus leading to a SYN/ACK war.

3.2. TCP self connects

Some systems have been found to be unable to process TCP connection requests in which the source endpoint {Source Address, Source Port} is the same as the destination end-point {Destination Address, Destination Port}. Such a scenario might arise e.g. if a process creates a socket, bind()s a local end-point (IP address and TCP port), and then issues a connect() to the same end-point as that specified to bind().

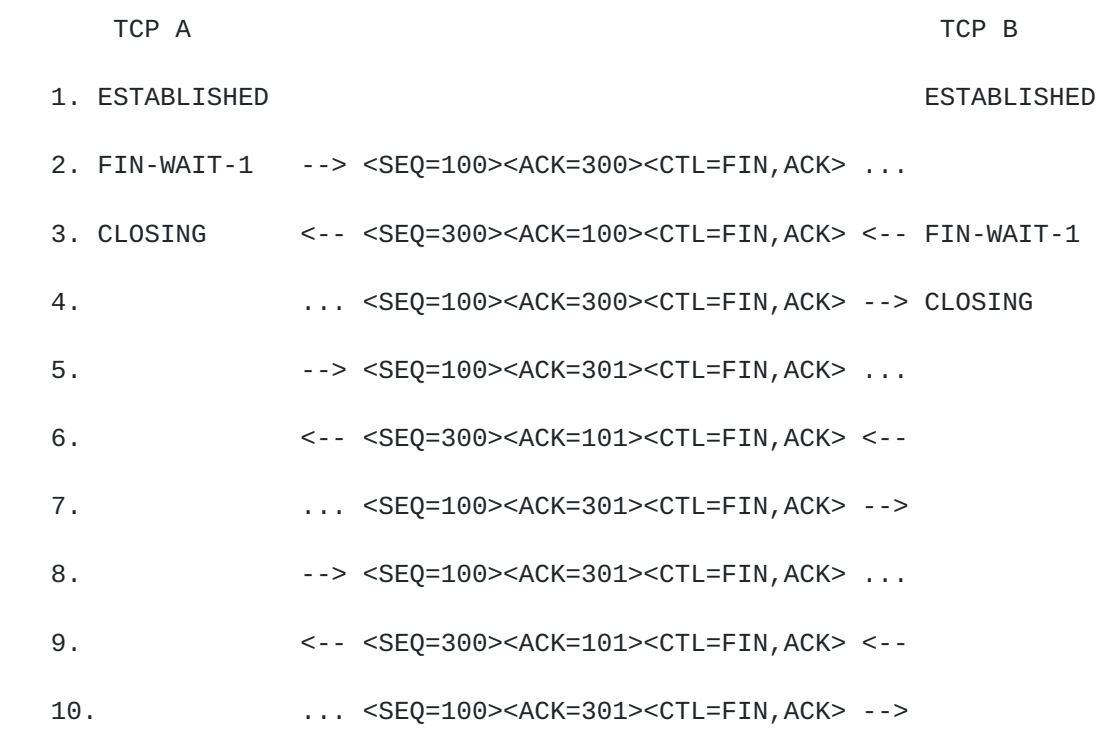
> While not widely employed in existing applications, such a socket could be employed as a "full-duplex pipe" for Inter-Process Communication (IPC).
>
> This scenario is described in detail in pp. 960-962 of [Wright1994].

The aforementioned scenario has been reported to cause malfunction of a number of implementations [CERT1996], and has been exploited in the past to perform Denial of Service (DoS) attacks [Meltman1997] [CPNI-TCP].

While this scenario is not common in the real world, TCP should nevertheless be able to process them without the need of any "extra" code: a SYN segment in which the source end-point {Source Address, Source Port} is the same as the destination end-point {Destination Address, Destination Port} should result in a "simultaneous open" scenario, such as the one described in page 32 of RFC 793 [RFC0793]. Therefore, those TCP implementations that correctly handle simultaneous opens should already be prepared to handle these unusual TCP segments.

3.3. TCP simultaneous close

The following figure illustrates a typical "simultaneous close" attempt, in which the FIN segments sent by each TCP end-point cross each other in the network.



(Failed) Simultaneous Connection Termination

In line 1, we assume that both end-points of the connection are in the ESTABLISHED state. In line 2, TCP A performs an "active close" by sending a FIN segment to TCP B, thus entering the FIN-WAIT-1 state. In line 3, TCP B performs an active close sending a FIN segment to TCP A, thus entering the FIN-WAIT-1 state; when this segment is processed by TCP A, it enters the CLOSING state (and its RCV.NXT becomes 301).

> Both FIN segments cross each other on the network, thus resulting in a "simultaneous connection termination" (or "simultaneous close") scenario.

In line 4, the FIN segment sent by TCP A arrives to TCP B, causing it to transition to the CLOSING state (at this point, TCP B's RCV.NXT becomes 101). In line 5, TCP A acknowledges the receipt of the TCP B's FIN segment, and also sets the FIN bit in the outgoing segment (since it has not yet been acknowledged). In line 6, TCP B acknowledges the receipt of TCP A's FIN segment, and also sets the FIN bit in the outgoing segment (since it has not yet been acknowledged). In line 7, the FIN/ACK from line 5 arrives at TCP B. In line 8, the FIN/ACK from line 6 fails the TCP sequence number validation check, and thus elicits a ACK segment (the segment also contains the FIN bit set, since it had not yet been acknowledged). In line 9, the FIN/ACK from line 7 fails the TCP sequence number

validation check, and hence elicits an ACK segment (the segment also contains the FIN bit set, since it had not yet been acknowledged). In line 10, the FIN/ACK from line 8 finally arrives at TCP B.

The packet exchange from lines 8-10 will repeat indefinitely, with both TCP end-points stuck in the CLOSING state, thus leading to a "FIN war": each FIN/ACK segment sent by a TCP will elicit a FIN/ACK from the other TCP, and each of these FIN/ACKs will in turn elicit more FIN/ACKs.

3.4. Simultaneous Window Probes

The following figure illustrates a scenario in which the "persist timer" at both TCP end-points expires, and both TCP end-points send a "window probes" that cross each other in the network.

```
   TCP A                                                TCP B

1. ESTABLISHED                                          ESTABLISHED

2.                      (both TCP windows open)

3.            --> <SEQ=100><DATA=1><ACK=300><CTL=ACK> ...

4.            <-- <SEQ=300><DATA=1><ACK=100><CTL=ACK> <--

5.            ... <SEQ=100><DATA=1><ACK=300><CTL=ACK> -->

6.            --> <SEQ=100><ACK=301><CTL=ACK>         ...

7.            <-- <SEQ=300><ACK=101><CTL=ACK>         <--

8.            ... <SEQ=100><ACK=301><CTL=ACK>         -->

9.            --> <SEQ=100><ACK=301><CTL=ACK>         ...

10.           <-- <SEQ=300><ACK=101><CTL=ACK>         <--

11.           ... <SEQ=100><ACK=301><CTL=ACK>         -->

              (Failed) Simultaneous Connection Termination
```

In line 1, we assume that both end-points of the connection are in the ESTABLISHED state; additionally, TCP A's RCV.NXT is 300, while TCP B's RCV.NXT is 100, and the receive window (RCV.WND) at both TCP end-points is 0. In line 2, both TCP windows open. In line 3, the "persist timer" at TCP A expires, and hence TCP A sends a "Window

Probe". In line 4, the "persist timer" at TCP B expires, and hence TCP B sends a "Window Probe".

Both Window Probes cross each other in the network.

When this probe arrives at TCP A, TCP a's RCV.NXT becomes 301, and an ACK segment is sent to advertise the new window (this ACK is shown in line 6). In line 5, TCP A's Window Probe from line 3 arrives at TCP B. TCP B's RCV-WND becomes 101. In line 6, TCP A sends the ACK to advertise the new window. In line 7, TCP B sends an ACK to advertise the new Window. When this ACK arrives at TCP A, the TCP Sequence Number validation fails, since SEG.SEQ=300 and RCV.NXT=301. Therefore, this segment elicits a new ACK (meant to re-synchronize the sequence numbers). In line 8, the ACK from line 6 arrives at TCP B. The TCP sequence number validation for this segment fails, since SEG.SEQ=100 AND RCV.NXT=101. Therefore, this segment elicits a new ACK (meant to re-synchronize the sequence numbers).

Line 9 and line 11 shows the ACK elicited by the segment from line 7, while line 10 shows the ACK elicited by the segment from line 8. The sequence numbers of these ACK segments will be considered invalid, and hence will elicit further ACKs. Therefore, the segment exchange from lines 9-11 will repeat indefinitely, thus leading to an "ACK war".

4. Updating RFC 793

4.1. TCP sequence number validation

The following text from Section 3.3 (pp. 25-26) of [RFC0793]:

---------- cut here ---------- cut here ----------

A segment is judged to occupy a portion of valid receive sequence space if

RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

or

RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

---------- cut here ---------- cut here ----------

is replaced with:

 ---------------- cut here -------------- cut here ----------------
A segment is judged to occupy a portion of valid receive sequence
space if

  RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND

or

  RCV.NXT-1 =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

The first part of this test checks to see if the beginning of the
segment falls in the window (or one byte to the left to the window),
the second part of the test checks to see if the end of the segment
falls in the window (or one byte to the left of the window); if the
segment passes either part of the test it contains data in the
window or control information that needs to be processed by TCP.

Actually, it is a little more complicated than this. Due to zero
windows and zero length segments, we have four cases for the
acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	RCV.NXT-1 =< SEG.SEQ <= RCV.NXT
0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT-1 =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

 ---------------- cut here -------------- cut here ----------------

Additionally, the following text from Section 3.9 (pp.69-70) of
[RFC0793]:

---------- cut here ---------- cut here ----------

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers may be held for later processing.

---------- cut here ---------- cut here ----------

is replaced with:

---------- cut here ---------- cut here ----------

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed. Acknowledgement information must still be processed when the contents of the incoming segment are one byte to the left of the receive window.

This is to handle simultaneous opens, simultaneous closes, and simultaneous window probes.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	RCV.NXT-1 =< SEG.SEQ <= RCV.NXT
0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT-1 =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN). Segments with higher beginning sequence numbers may be held for later processing. Acknowledgement information must still be processed when the contents of the incoming segment are

```
      one byte to the left of the receive window.
 ---------------- cut here -------------- cut here ----------------
```

4.2. Alternative fix for TCP sequence number validation

The Linux kernel performs a slightly different TCP sequence number validation check, that can accommodate window probes of any size (as opposed to the de facto standard 1-byte window probes). This makes the code more general, at the expense of additional state in the TCB (e.g., the TCP sequence number employed in the last window probe).

4.3. TCP self connects

TCP MUST be able to gracefully handle connection requests (i.e., SYN segments) in which the source end-point (IP Source Address, TCP Source Port) is the same as the destination end-point (IP Destination Address, TCP Destination Port). Such segments MUST result in a TCP "simultaneous open", such as the one described in page 32 of RFC 793 [RFC0793].

> Those TCP implementations that correctly handle simultaneous opens are expected to gracefully handle this scenario.

5. IANA Considerations

This document has no IANA actions. The RFC Editor is requested to remove this section before publishing this document as an RFC.

6. Security Considerations

This document describes a problem found in the current validation rules for TCP sequence numbers. The aforementioned problem has affected some popular TCP implementations, typically leading to connection failures, system crashes, or other undesirable behaviors. This document formally updates RFC 793, such that the aforementioned issues are eliminated.

7. Acknowledgements

Thhe authors of this document would like to thank Theo de Raadt, Rui Paulo and Michael Scharf for providing valuable comments on earlier versions of this document.

8. References

8.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[CERT1996] CERT, "CERT Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks", 1996, <http://www.cert.org/advisories/CA-1996-21.html>.

[CPNI-TCP] Gont, F., "CPNI Technical Note 3/2009: Security Assessment of the Transmission Control Protocol (TCP)", 2009, <http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>.

[Meltman1997] Meltman, "new TCP/IP bug in win95. Post to the bugtraq mailing-list", 1996, <http://insecure.org/sploits/land.ip.DOS.html>.

[Wright1994] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1994.

Authors' Addresses

Fernando Gont
UTN-FRH / SI6 Networks
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: http://www.si6networks.com

David Borman
Quantum Corporation
1155 Centre Pointe Drive, Suite 1
Mendota Heights, MN 55120
U.S.A.

Phone: 651-688-4394
Email: david.borman@quantum.com