TCP Problems with Path MTU Discovery

Status of this Memo

Copyright Notice

Abstract

   This memo catalogs several known Transmission Control Protocol (TCP)
   implementation problems dealing with Path Maximum Transmission Unit
   Discovery (PMTUD), including the long-standing black hole problem,
   stretch acknowlegements (ACKs) due to confusion between Maximum
   Segment Size (MSS) and segment size, and MSS advertisement based on
   PMTU.

1. Introduction

   This memo catalogs several known TCP implementation problems dealing
   with Path MTU Discovery [RFC1191], including the long-standing black
   hole problem, stretch ACKs due to confusion between MSS and segment
   size, and MSS advertisement based on PMTU. The goal in doing so is
   to improve conditions in the existing Internet by enhancing the
   quality of current TCP/IP implementations.

   While Path MTU Discovery (PMTUD) can be used with any upper-layer
   protocol, it is most commonly used by TCP; this document does not
   attempt to treat problems encountered by other upper-layer protocols.
   Path MTU Discovery for IPv6 [RFC1981] treats only IPv6-dependent
   issues, but not the TCP issues brought up in this document.

   Each problem is defined as follows:

   Name of Problem
      The name associated with the problem. In this memo, the name is
      given as a subsection heading.

Classification
   One or more problem categories for which the problem is
   classified: "congestion control", "performance", "reliability",
   "non-interoperation -- connectivity failure".

Description
   A definition of the problem, succinct but including necessary
   background material.

Significance
   A brief summary of the sorts of environments for which the problem
   is significant.

Implications
   Why the problem is viewed as a problem.

Relevant RFCs
   The RFCs defining the TCP specification with which the problem
   conflicts. These RFCs often qualify behavior using terms such as
   MUST, SHOULD, MAY, and others written capitalized. See RFC 2119
   for the exact interpretation of these terms.

Trace file demonstrating the problem
   One or more ASCII trace files demonstrating the problem, if
   applicable.

Trace file demonstrating correct behavior
   One or more examples of how correct behavior appears in a trace,
   if applicable.

References
   References that further discuss the problem.

How to detect
   How to test an implementation to see if it exhibits the problem.
   This discussion may include difficulties and subtleties associated
   with causing the problem to manifest itself, and with interpreting
   traces to detect the presence of the problem (if applicable).

How to fix
   For known causes of the problem, how to correct the
   implementation.

2. Known implementation problems

2.1.

   Name of Problem
      Black Hole Detection

   Classification
      Non-interoperation -- connectivity failure

   Description
      A host performs Path MTU Discovery by sending out as large a
      packet as possible, with the Don't Fragment (DF) bit set in the IP
      header. If the packet is too large for a router to forward on to
      a particular link, the router must send an ICMP Destination
      Unreachable -- Fragmentation Needed message to the source address.
      The host then adjusts the packet size based on the ICMP message.

      As was pointed out in [RFC1435], routers don't always do this
      correctly -- many routers fail to send the ICMP messages, for a
      variety of reasons ranging from kernel bugs to configuration
      problems. Firewalls are often misconfigured to suppress all ICMP
      messages. IPsec [RFC2401] and IP-in-IP [RFC2003] tunnels
      shouldn't cause these sorts of problems, if the implementations
      follow the advice in the appropriate documents.

      PMTUD, as documented in [RFC1191], fails when the appropriate ICMP
      messages are not received by the originating host. The upper-
      layer protocol continues to try to send large packets and, without
      the ICMP messages, never discovers that it needs to reduce the
      size of those packets. Its packets are disappearing into a PMTUD
      black hole.

   Significance
      When PMTUD fails due to the lack of ICMP messages, TCP will also
      completely fail under some conditions.

   Implications
      This failure is especially difficult to debug, as pings and some
      interactive TCP connections to the destination host work. Bulk
      transfers fail with the first large packet and the connection
      eventually times out.

      These situations can almost always be blamed on a misconfiguration
      within the network, which should be corrected. However it seems
      inappropriate for some TCP implementations to suffer

      interoperability failures over paths which do not affect other TCP
      implementations (i.e. those without PMTUD). This creates a market
      disincentive for deploying TCP implementation with PMTUD enabled.

   Relevant RFCs
      RFC 1191 describes Path MTU Discovery. RFC 1435 provides an early
      description of these sorts of problems.

   Trace file demonstrating the problem
      Made using tcpdump [Jacobson89] recording at an intermediate host.

      20:12:11.951321 A > B: S 1748427200:1748427200(0)
           win 49152 <mss 1460>
      20:12:11.951829 B > A: S 1001927984:1001927984(0)
           ack 1748427201 win 16384 <mss 65240>
      20:12:11.955230 A > B: . ack 1 win 49152 (DF)
      20:12:11.959099 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:12:13.139074 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:12:16.188685 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:12:22.290483 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:12:34.491856 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:12:58.896405 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:13:47.703184 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:14:52.780640 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:15:57.856037 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:17:02.932431 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:18:08.009337 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:19:13.090521 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:20:18.168066 A > B: . 1:1461(1460) ack 1 win 49152 (DF)
      20:21:23.242761 A > B: R 1461:1461(0) ack 1 win 49152 (DF)

      The short SYN packet has no trouble traversing the network, due to
      its small size. Similarly, ICMP echo packets used to diagnose
      connectivity problems will succeed.

      Large data packets fail to traverse the network. Eventually the
      connection times out. This can be especially confusing when the
      application starts out with a very small write, which succeeds,
      following up with many large writes, which then fail.

   Trace file demonstrating correct behavior

      Made using tcpdump recording at an intermediate host.

      16:48:42.659115 A > B: S 271394446:271394446(0)
           win 8192 <mss 1460> (DF)
      16:48:42.672279 B > A: S 2837734676:2837734676(0)
           ack 271394447 win 16384 <mss 65240>

```
16:48:42.676890 A > B: . ack 1 win 8760 (DF)
16:48:42.870574 A > B: . 1:1461(1460) ack 1 win 8760 (DF)
16:48:42.871799 A > B: . 1461:2921(1460) ack 1 win 8760 (DF)
16:48:45.786814 A > B: . 1:1461(1460) ack 1 win 8760 (DF)
16:48:51.794676 A > B: . 1:1461(1460) ack 1 win 8760 (DF)
16:49:03.808912 A > B: . 1:537(536) ack 1 win 8760
16:49:04.016476 B > A: . ack 537 win 16384
16:49:04.021245 A > B: . 537:1073(536) ack 1 win 8760
16:49:04.021697 A > B: . 1073:1609(536) ack 1 win 8760
16:49:04.120694 B > A: . ack 1609 win 16384
16:49:04.126142 A > B: . 1609:2145(536) ack 1 win 8760
```

   In this case, the sender sees four packets fail to traverse the
   network (using a two-packet initial send window) and turns off
   PMTUD. All subsequent packets have the DF flag turned off, and
   the size set to the default value of 536 [RFC1122].

References
   This problem has been discussed extensively on the tcp-impl
   mailing list; the name "black hole" has been in use for many
   years.

How to detect
   This shows up as a TCP connection which hangs (fails to make
   progress) until closed by timeout (this often manifests itself as
   a connection that connects and starts to transfer, then eventually
   terminates after 15 minutes with zero bytes transfered). This is
   particularly annoying with an application like ftp, which will
   work perfectly while it uses small packets for control
   information, and then fail on bulk transfers.

   A series of ICMP echo packets will show that the two end hosts are
   still capable of passing packets, a series of MTU-sized ICMP echo
   packets will show some fragmentation, and a series of MTU-sized
   ICMP echo packets with DF set will fail. This can be confusing
   for network engineers trying to diagnose the problem.

   There are several traceroute implementations that do PMTUD, and
   can demonstrate the problem.

How to fix
   TCP should notice that the connection is timing out. After
   several timeouts, TCP should attempt to send smaller packets,
   perhaps turning off the DF flag for each packet. If this
   succeeds, it should continue to turn off PMTUD for the connection
   for some reasonable period of time, after which it should probe
   again to try to determine if the path has changed.

      Note that, under IPv6, there is no DF bit -- it is implicitly on
      at all times. Fragmentation is not allowed in routers, only at
      the originating host. Fortunately, the minimum supported MTU for
      IPv6 is 1280 octets, which is significantly larger than the 68
      octet minimum in IPv4. This should make it more reasonable for
      IPv6 TCP implementations to fall back to 1280 octet packets, when
      IPv4 implementations will probably have to turn off DF to respond
      to black hole detection.

      Ideally, the ICMP black holes should be fixed when they are found.

      If hosts start to implement black hole detection, it may be that
      these problems will go unnoticed and unfixed. This is especially
      unfortunate, since detection can take several seconds each time,
      and these delays could result in a significant, hidden degradation
      of performance. Hosts that implement black hole detection should
      probably log detected black holes, so that they can be fixed.

2.2.

   Name of Problem
      Stretch ACK due to PMTUD

   Classification
      Congestion Control / Performance

   Description
      When a naively implemented TCP stack communicates with a PMTUD
      equipped stack, it will try to generate an ACK for every second
      full-sized segment. If it determines the full-sized segment based
      on the advertised MSS, this can degrade badly in the face of
      PMTUD.

      The PMTU can wind up being a small fraction of the advertised MSS;
      in this case, an ACK would be generated only very infrequently.

   Significance

      Stretch ACKs have a variety of unfortunate effects, more fully
      outlined in [RFC2525]. Most of these have to do with encouraging
      a more bursty connection, due to the infrequent arrival of ACKs.
      They can also impede congestion window growth.

   Implications

      The complete implications of stretch ACKs are outlined in
      [RFC2525].

   Relevant RFCs
      RFC 1122 outlines the requirements for frequency of ACK
      generation. [RFC2581] expands on this and clarifies that delayed
      ACK is a SHOULD, not a MUST.

   Trace file demonstrating it

      Made using tcpdump recording at an intermediate host. The
      timestamp options from all but the first two packets have been
      removed for clarity.

   18:16:52.976657 A > B: S 3183102292:3183102292(0) win 16384
        <mss 4312,nop,wscale 0,nop,nop,timestamp 12128 0> (DF)
   18:16:52.979580 B > A: S 2022212745:2022212745(0) ack 3183102293 win
        49152 <mss 4312,nop,wscale 1,nop,nop,timestamp 1592957 12128> (DF)
   18:16:52.979738 A > B: . ack 1 win 17248 (DF)
   18:16:52.982473 A > B: . 1:4301(4300) ack 1 win 17248 (DF)
   18:16:52.982557 C > A: icmp: B unreachable -
        need to frag (mtu 1500)! (DF)
   18:16:52.985839 B > A: . ack 1 win 32768 (DF)
   18:16:54.129928 A > B: . 1:1449(1448) ack 1 win 17248 (DF)
        .
        .
        .
   18:16:58.507078 A > B: . 1463941:1465389(1448) ack 1 win 17248   (DF)
   18:16:58.507200 A > B: . 1465389:1466837(1448) ack 1 win 17248   (DF)
   18:16:58.507326 A > B: . 1466837:1468285(1448) ack 1 win 17248   (DF)
   18:16:58.507439 A > B: . 1468285:1469733(1448) ack 1 win 17248   (DF)
   18:16:58.524763 B > A: . ack 1452357 win 32768   (DF)
   18:16:58.524986 B > A: . ack 1461045 win 32768   (DF)
   18:16:58.525138 A > B: . 1469733:1471181(1448) ack 1 win 17248   (DF)
   18:16:58.525268 A > B: . 1471181:1472629(1448) ack 1 win 17248   (DF)
   18:16:58.525393 A > B: . 1472629:1474077(1448) ack 1 win 17248   (DF)
   18:16:58.525516 A > B: . 1474077:1475525(1448) ack 1 win 17248   (DF)
   18:16:58.525642 A > B: . 1475525:1476973(1448) ack 1 win 17248   (DF)
   18:16:58.525766 A > B: . 1476973:1478421(1448) ack 1 win 17248   (DF)
   18:16:58.526063 A > B: . 1478421:1479869(1448) ack 1 win 17248   (DF)
   18:16:58.526187 A > B: . 1479869:1481317(1448) ack 1 win 17248   (DF)
   18:16:58.526310 A > B: . 1481317:1482765(1448) ack 1 win 17248   (DF)
   18:16:58.526432 A > B: . 1482765:1484213(1448) ack 1 win 17248   (DF)
   18:16:58.526561 A > B: . 1484213:1485661(1448) ack 1 win 17248   (DF)
   18:16:58.526671 A > B: . 1485661:1487109(1448) ack 1 win 17248   (DF)
   18:16:58.537944 B > A: . ack 1478421 win 32768   (DF)
   18:16:58.538328 A > B: . 1487109:1488557(1448) ack 1 win 17248 (DF)

   Note that the interval between ACKs is significantly larger than two
   times the segment size; it works out to be almost exactly two times
   the advertised MSS. This transfer was long enough that it could be
   verified that the stretch ACK was not the result of lost ACK packets.

   Trace file demonstrating correct behavior

   Made using tcpdump recording at an intermediate host. The timestamp
   options from all but the first two packets have been removed for
   clarity.

   18:13:32.287965 A > B: S 2972697496:2972697496(0)
        win 16384 <mss 4312,nop,wscale 0,nop,nop,timestamp 11326 0> (DF)
   18:13:32.290785 B > A: S 245639054:245639054(0)
        ack 2972697497 win 34496 <mss 4312> (DF)
   18:13:32.290941 A > B: . ack 1 win 17248 (DF)
   18:13:32.293774 A > B: . 1:4313(4312) ack 1 win 17248 (DF)
   18:13:32.293856 C > A: icmp: B unreachable -
        need to frag (mtu 1500)! (DF)
   18:13:33.637338 A > B: . 1:1461(1460) ack 1 win 17248 (DF)
        .
        .
        .
   18:13:35.561691 A > B: . 1514021:1515481(1460) ack 1 win 17248 (DF)
   18:13:35.561814 A > B: . 1515481:1516941(1460) ack 1 win 17248 (DF)
   18:13:35.561938 A > B: . 1516941:1518401(1460) ack 1 win 17248 (DF)
   18:13:35.562059 A > B: . 1518401:1519861(1460) ack 1 win 17248 (DF)
   18:13:35.562174 A > B: . 1519861:1521321(1460) ack 1 win 17248 (DF)
   18:13:35.564008 B > A: . ack 1481901 win 64680 (DF)
   18:13:35.564383 A > B: . 1521321:1522781(1460) ack 1 win 17248 (DF)
   18:13:35.564499 A > B: . 1522781:1524241(1460) ack 1 win 17248 (DF)
   18:13:35.615576 B > A: . ack 1484821 win 64680 (DF)
   18:13:35.615646 B > A: . ack 1487741 win 64680 (DF)
   18:13:35.615716 B > A: . ack 1490661 win 64680 (DF)
   18:13:35.615784 B > A: . ack 1493581 win 64680 (DF)
   18:13:35.615856 B > A: . ack 1496501 win 64680 (DF)
   18:13:35.615952 A > B: . 1524241:1525701(1460) ack 1 win 17248 (DF)
   18:13:35.615966 B > A: . ack 1499421 win 64680 (DF)
   18:13:35.616088 A > B: . 1525701:1527161(1460) ack 1 win 17248 (DF)
   18:13:35.616105 B > A: . ack 1502341 win 64680 (DF)
   18:13:35.616211 A > B: . 1527161:1528621(1460) ack 1 win 17248 (DF)
   18:13:35.616228 B > A: . ack 1505261 win 64680 (DF)
   18:13:35.616327 A > B: . 1528621:1530081(1460) ack 1 win 17248 (DF)
   18:13:35.616349 B > A: . ack 1508181 win 64680 (DF)
   18:13:35.616448 A > B: . 1530081:1531541(1460) ack 1 win 17248 (DF)
   18:13:35.616565 A > B: . 1531541:1533001(1460) ack 1 win 17248 (DF)
   18:13:35.616891 A > B: . 1533001:1534461(1460) ack 1 win 17248 (DF)

In this trace, an ACK is generated for every two segments that
arrive. (The segment size is slightly larger in this trace, even
though the source hosts are the same, because of the lack of
timestamp options in this trace.)

How to detect
This condition can be observed in a packet trace when the advertised
MSS is significantly larger than the actual PMTU of a connection.

How to fix Several solutions for this problem have been proposed:

A simple solution is to ACK every other packet, regardless of size.
This has the drawback of generating large numbers of ACKs in the face
of lots of very small packets; this shows up with applications like
the X Window System.

A slightly more complex solution would monitor the size of incoming
segments and try to determine what segment size the sender is using.
This requires slightly more state in the receiver, but has the
advantage of making receiver silly window syndrome avoidance
computations more accurate [RFC813].

2.3.

   Name of Problem
   Determining MSS from PMTU

   Classification
   Performance

   Description
   The MSS advertised at the start of a connection should be based on
   the MTU of the interfaces on the system. (For efficiency and other
   reasons this may not be the largest MSS possible.) Some systems use
   PMTUD determined values to determine the MSS to advertise.

   This results in an advertised MSS that is smaller than the largest
   MTU the system can receive.

   Significance
   The advertised MSS is an indication to the remote system about the
   largest TCP segment that can be received [RFC879]. If this value is
   too small, the remote system will be forced to use a smaller segment
   size when sending, purely because the local system found a particular
   PMTU earlier.

Given the asymmetric nature of many routes on the Internet
[Paxson97], it seems entirely possible that the return PMTU is
different from the sending PMTU. Limiting the segment size in this
way can reduce performance and frustrate the PMTUD algorithm.

Even if the route was symmetric, setting this artificially lowered
limit on segment size will make it impossible to probe later to
determine if the PMTU has changed.

Implications
The whole point of PMTUD is to send as large a segment as possible.
If long-running connections cannot successfully probe for larger
PMTU, then potential performance gains will be impossible to realize.
This destroys the whole point of PMTUD.

Relevant RFCs RFC 1191. [RFC879] provides a complete discussion of
MSS calculations and appropriate values. Note that this practice
does not violate any of the specifications in these RFCs.

Trace file demonstrating it
This trace was made using tcpdump running on an intermediate host.
Host A initiates two separate consecutive connections, A1 and A2, to
host B. Router C is the location of the MTU bottleneck. As usual,
TCP options are removed from all non-SYN packets.

```
22:33:32.305912 A1 > B: S 1523306220:1523306220(0)
     win 8760 <mss 1460> (DF)
22:33:32.306518 B > A1: S 729966260:729966260(0)
     ack 1523306221 win 16384 <mss 65240>
22:33:32.310307 A1 > B: . ack 1 win 8760 (DF)
22:33:32.323496 A1 > B: P 1:1461(1460) ack 1 win 8760 (DF)
22:33:32.323569 C > A1: icmp: 129.99.238.5 unreachable -
     need to frag (mtu 1024) (DF) (ttl 255, id 20666)
22:33:32.783694 A1 > B: . 1:985(984) ack 1 win 8856 (DF)
22:33:32.840817 B > A1: . ack 985 win 16384
22:33:32.845651 A1 > B: . 1461:2445(984) ack 1 win 8856 (DF)
22:33:32.846094 B > A1: . ack 985 win 16384
22:33:33.724392 A1 > B: . 985:1969(984) ack 1 win 8856 (DF)
22:33:33.724893 B > A1: . ack 2445 win 14924
22:33:33.728591 A1 > B: . 2445:2921(476) ack 1 win 8856 (DF)
22:33:33.729161 A1 > B: . ack 1 win 8856 (DF)
22:33:33.840758 B > A1: . ack 2921 win 16384

[...]

22:33:34.238659 A1 > B: F 7301:8193(892) ack 1 win 8856 (DF)
22:33:34.239036 B > A1: . ack 8194 win 15492
22:33:34.239303 B > A1: F 1:1(0) ack 8194 win 16384
```

```
22:33:34.242971 A1 > B: . ack 2 win 8856 (DF)
22:33:34.454218 A2 > B: S 1523591299:1523591299(0)
     win 8856 <mss 984> (DF)
22:33:34.454617 B > A2: S 732408874:732408874(0)
     ack 1523591300 win 16384 <mss 65240>
22:33:34.457516 A2 > B: . ack 1 win 8856 (DF)
22:33:34.470683 A2 > B: P 1:985(984) ack 1 win 8856 (DF)
22:33:34.471144 B > A2: . ack 985 win 16384
22:33:34.476554 A2 > B: . 985:1969(984) ack 1 win 8856 (DF)
22:33:34.477580 A2 > B: P 1969:2953(984) ack 1 win 8856 (DF)


[...]
```

Notice that the SYN packet for session A2 specifies an MSS of 984.

Trace file demonstrating correct behavior

As before, this trace was made using tcpdump running on an
intermediate host. Host A initiates two separate consecutive
connections, A1 and A2, to host B. Router C is the location of the
MTU bottleneck. As usual, TCP options are removed from all non-SYN
packets.

```
22:36:58.828602 A1 > B: S 3402991286:3402991286(0) win 32768
     <mss 4312,wscale 0,nop,timestamp 1123370309 0,
      echo 1123370309> (DF)
22:36:58.844040 B > A1: S 946999880:946999880(0)
     ack 3402991287 win 16384
     <mss 65240,nop,wscale 0,nop,nop,timestamp 429552 1123370309>
22:36:58.848058 A1 > B: . ack 1 win 32768  (DF)
22:36:58.851514 A1 > B: P 1:1025(1024) ack 1 win 32768  (DF)
22:36:58.851584 C > A1: icmp: 129.99.238.5 unreachable -
     need to frag (mtu 1024) (DF)
22:36:58.855885 A1 > B: . 1:969(968) ack 1 win 32768  (DF)
22:36:58.856378 A1 > B: . 969:985(16) ack 1 win 32768  (DF)
22:36:59.036309 B > A1: . ack 985 win 16384
22:36:59.039255 A1 > B: FP 985:1025(40) ack 1 win 32768  (DF)
22:36:59.039623 B > A1: . ack 1026 win 16344
22:36:59.039828 B > A1: F 1:1(0) ack 1026 win 16384
22:36:59.043037 A1 > B: . ack 2 win 32768  (DF)
22:37:01.436032 A2 > B: S 3404812097:3404812097(0) win 32768
     <mss 4312,wscale 0,nop,timestamp 1123372916 0,
      echo 1123372916> (DF)
22:37:01.436424 B > A2: S 949814769:949814769(0)
     ack 3404812098 win 16384
     <mss 65240,nop,wscale 0,nop,nop,timestamp 429562 1123372916>
22:37:01.440147 A2 > B: . ack 1 win 32768  (DF)
22:37:01.442736 A2 > B: . 1:969(968) ack 1 win 32768  (DF)
```

```
22:37:01.442894 A2 > B: P 969:985(16) ack 1 win 32768  (DF)
22:37:01.443283 B > A2: . ack 985 win 16384
22:37:01.446068 A2 > B: P 985:1025(40) ack 1 win 32768  (DF)
22:37:01.446519 B > A2: . ack 1025 win 16384
22:37:01.448465 A2 > B: F 1025:1025(0) ack 1 win 32768  (DF)
22:37:01.448837 B > A2: . ack 1026 win 16384
22:37:01.449007 B > A2: F 1:1(0) ack 1026 win 16384
22:37:01.452201 A2 > B: . ack 2 win 32768  (DF)
```

   Note that the same MSS was used for both session A1 and session A2.

   How to detect
   This can be detected using a packet trace of two separate
   connections; the first should invoke PMTUD; the second should start
   soon enough after the first that the PMTU value does not time out.

   How to fix
   The MSS should be determined based on the MTUs of the interfaces on
   the system, as outlined in [RFC1122] and [RFC1191].

3. Security Considerations

   The one security concern raised by this memo is that ICMP black holes
   are often caused by over-zealous security administrators who block
   all ICMP messages. It is vitally important that those who design and
   deploy security systems understand the impact of strict filtering on
   upper-layer protocols. The safest web site in the world is worthless
   if most TCP implementations cannot transfer data from it. It would
   be far nicer to have all of the black holes fixed rather than fixing
   all of the TCP implementations.

4. Acknowledgements

   Thanks to Mark Allman, Vern Paxson, and Jamshid Mahdavi for generous
   help reviewing the document, and to Matt Mathis for early suggestions
   of various mechanisms that can cause PMTUD black holes, as well as
   review. The structure for describing TCP problems, and the early
   description of that structure is from [RFC2525]. Special thanks to
   Amy Bock, who helped perform the PMTUD tests which discovered these
   bugs.

5. References

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

   [RFC1122]   Braden, R., "Requirements for Internet Hosts --
                Communication Layers", STD 3, RFC 1122, October 1989.

   [RFC813]   Clark, D., "Window and Acknowledgement Strategy in TCP",
                RFC 813, July 1982.

   [Jacobson89] V. Jacobson, C. Leres, and S. McCanne, tcpdump, June
                1989, ftp.ee.lbl.gov

   [RFC1435]   Knowles, S., "IESG Advice from Experience with Path MTU
                Discovery", RFC 1435, March 1993.

   [RFC1191]   Mogul, J. and S. Deering, "Path MTU discovery", RFC
                1191, November 1990.

   [RFC1981]   McCann, J., Deering, S. and J. Mogul, "Path MTU
                Discovery for IP version 6", RFC 1981, August 1996.

   [Paxson96]   V. Paxson, "End-to-End Routing Behavior in the
                Internet", IEEE/ACM Transactions on Networking (5),
                pp.~601-615, Oct. 1997.

   [RFC2525]   Paxon, V., Allman, M., Dawson, S., Fenner, W., Griner,
                J., Heavens, I., Lahey, K., Semke, I. and B. Volz,
                "Known TCP Implementation Problems", RFC 2525, March
                1999.

   [RFC879]   Postel, J., "The TCP Maximum Segment Size and Related
                Topics", RFC 879, November 1983.

   [RFC2001]   Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast
                Retransmit, and Fast Recovery Algorithms", RFC 2001,
                January 1997.

6. Author's Address

    Kevin Lahey
    dotRocket, Inc.
    1901 S. Bascom Ave., Suite 300
    Campbell, CA 95008
    USA

    Phone: +1 408-371-8977 x115
    email: kml@dotrocket.com

7. Full Copyright Statement

Acknowledgement