Architectural Guidelines for Multipath TCP Development

Abstract

   Hosts are often connected by multiple paths, but TCP restricts
   communications to a single path per transport connection. Resource
   usage within the network would be more efficient were these multiple
   paths able to be used concurrently. This should enhance user
   experience through improved resilience to network failure and higher
   throughput.

   This document outlines architectural guidelines for the development
   of a Multipath Transport Protocol, with references to how these
   architectural components come together in the development of a
   Multipath TCP (MPTCP). This document lists certain high-level design
   decisions that provide foundations for the design of the MPTCP
   protocol, based upon these architectural requirements.

Status of This Memo

   This document is not an Internet Standards Track specification; it is
   published for informational purposes.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Not all documents
   approved by the IESG are a candidate for any level of Internet
   Standard; see Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc6182.

Copyright Notice

Table of Contents

1. Introduction

   As the Internet evolves, demands on Internet resources are ever-
   increasing, but often these resources (in particular, bandwidth)
   cannot be fully utilized due to protocol constraints both on the end-
   systems and within the network. If these resources could be used
   concurrently, end user experience could be greatly improved. Such
   enhancements would also reduce the necessary expenditure on network
   infrastructure that would otherwise be needed to create an equivalent
   improvement in user experience. By the application of resource
   pooling [3], these available resources can be 'pooled' such that they
   appear as a single logical resource to the user.

   Multipath transport aims to realize some of the goals of resource
   pooling by simultaneously making use of multiple disjoint (or
   partially disjoint) paths across a network. The two key benefits of
   multipath transport are the following:

   o To increase the resilience of the connectivity by providing
      multiple paths, protecting end hosts from the failure of one.

   o To increase the efficiency of the resource usage, and thus
      increase the network capacity available to end hosts.

   Multipath TCP is a modified version of TCP [1] that implements a
   multipath transport and achieves these goals by pooling multiple
   paths within a transport connection, transparently to the
   application. Multipath TCP is primarily concerned with utilizing
   multiple paths end-to-end, where one or both of the end hosts are
   multihomed. It may also have applications where multiple paths exist
   within the network and can be manipulated by an end host, such as
   using different port numbers with Equal Cost MultiPath (ECMP) [4].

   MPTCP, defined in [5], is a specific protocol that instantiates the
   Multipath TCP concept. This document looks both at general
   architectural principles for a Multipath TCP fulfilling the goals
   described in Section 2, as well as the key design decisions behind
   MPTCP, which are detailed in Section 5.

   Although multihoming and multipath functions are not new to transport
   protocols (Stream Control Transmission Protocol (SCTP) [6] being a
   notable example), MPTCP aims to gain wide-scale deployment by
   recognizing the importance of application and network compatibility
   goals. These goals, discussed in detail in Section 2, relate to the
   appearance of MPTCP to the network (so non-MPTCP-aware entities see
   it as TCP) and to the application (through providing a service
   equivalent to TCP for non-MPTCP-aware applications).

This document has three key purposes: (i) it describes goals for a
multipath transport -- goals that MPTCP is designed to meet; (ii) it
lays out an architectural basis for MPTCP's design -- a discussion
that applies to other multipath transports as well; and (iii) it
discusses and documents high-level design decisions made in MPTCP's
development, and considers their implications.

Companion documents to this architectural overview are those that
provide details of the protocol extensions [5], congestion control
algorithms [7], and application-level considerations [8]. Put
together, these components specify a complete Multipath TCP design.
Note that specific components are replaceable in accordance with the
layer and functional decompositions discussed in this document.

1.1. Requirements Language

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 [2].

1.2. Terminology

Regular/Single-Path TCP: The standard version of TCP [1] in use
   today, operating between a single pair of IP addresses and ports.

Multipath TCP: A modified version of the TCP protocol that supports
   the simultaneous use of multiple paths between hosts.

Path: A sequence of links between a sender and a receiver, defined
   in this context by a source and destination address pair.

Host: An end host either initiating or terminating a Multipath TCP
   connection.

MPTCP: The proposed protocol extensions specified in [5] to provide
   a Multipath TCP implementation.

Subflow: A flow of TCP segments operating over an individual path,
   which forms part of a larger Multipath TCP connection.

(Multipath TCP) Connection: A set of one or more subflows combined
   to provide a single Multipath TCP service to an application at a
   host.

1.3. Reference Scenario

   The diagram shown in Figure 1 illustrates a typical usage scenario
   for Multipath TCP. Two hosts, A and B, are communicating with each
   other. These hosts are multihomed and multi-addressed, providing two
   disjoint connections to the Internet. The addresses on each host are
   referred to as A1, A2, B1, and B2. There are therefore up to four
   different paths between the two hosts: A1-B1, A1-B2, A2-B1, A2-B2.

```
            +------+            _____            +------+
            |      |A1 _____ (          ) _____ B1|      |
            | Host |--/       (          )      \--| Host |
            |      |         (    Internet   )        |      |
            |   A  |--_____(          )_____/--|   B  |
            |      |A2        (_____)        B2|      |
            +------+                                +------+
```
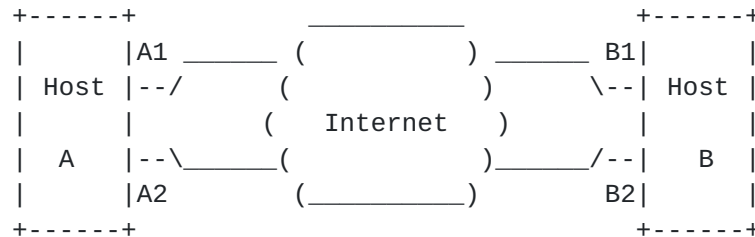
                 Figure 1: Simple Multipath TCP Usage Scenario

   The scenario could have any number of addresses (1 or more) on each
   host, as long as the number of paths available between the two hosts
   is 2 or more (i.e., num_addr(A) * num_addr(B) > 1). The paths
   created by these address combinations through the Internet need not
   be entirely disjoint -- potential fairness issues introduced by
   shared bottlenecks need to be handled by the Multipath TCP congestion
   controller. Furthermore, the paths through the Internet often do not
   provide a pure end-to-end service, and instead may be affected by
   middleboxes such as NATs and firewalls.

2. Goals

   This section outlines primary goals that Multipath TCP aims to meet.
   These are broadly broken down into the following: functional goals,
   which steer services and features that Multipath TCP must provide,
   and compatibility goals, which determine how Multipath TCP should
   appear to entities that interact with it.

2.1. Functional Goals

   In supporting the use of multiple paths, Multipath TCP has the
   following two functional goals.

   o Improve Throughput: Multipath TCP MUST support the concurrent use
      of multiple paths. To meet the minimum performance incentives for
      deployment, a Multipath TCP connection over multiple paths SHOULD
      achieve no worse throughput than a single TCP connection over the
      best constituent path.

   o Improve Resilience: Multipath TCP MUST support the use of multiple
      paths interchangeably for resilience purposes, by permitting
      segments to be sent and re-sent on any available path. It follows
      that, in the worst case, the protocol MUST be no less resilient
      than regular single-path TCP.

   As distribution of traffic among available paths and responses to
   congestion are done in accordance with resource pooling principles
   [3], a secondary effect of meeting these goals is that widespread use
   of Multipath TCP over the Internet should improve overall network
   utility by shifting load away from congested bottlenecks and by
   taking advantage of spare capacity wherever possible.

   Furthermore, Multipath TCP SHOULD feature automatic negotiation of
   its use. A host supporting Multipath TCP that requires the other
   host to do so too must be able to detect reliably whether this host
   does in fact support the required extensions, using them if so, and
   otherwise automatically falling back to single-path TCP.

2.2. Compatibility Goals

   In addition to the functional goals listed above, a Multipath TCP
   must meet a number of compatibility goals in order to support
   deployment in today's Internet. These goals fall into the following
   categories.

2.2.1. Application Compatibility

   Application compatibility refers to the appearance of Multipath TCP
   to the application both in terms of the API that can be used and the
   expected service model that is provided.

   Multipath TCP MUST follow the same service model as TCP [1]: in-
   order, reliable, and byte-oriented delivery. Furthermore, a
   Multipath TCP connection SHOULD provide the application with no worse
   throughput or resilience than it would expect from running a single
   TCP connection over any one of its available paths. A Multipath TCP
   may not, however, be able to provide the same level of consistency of
   throughput and latency as a single TCP connection. These, and other,
   application considerations are discussed in detail in [8].

   A multipath-capable equivalent of TCP MUST retain some level of
   backward compatibility with existing TCP APIs, so that existing
   applications can use the newer transport merely by upgrading the
   operating systems of the end hosts. This does not preclude the use
   of an advanced API to permit multipath-aware applications to specify

preferences, nor for users to configure their systems in a different
way from the default, for example switching on or off the automatic
use of multipath extensions.

It is possible for regular TCP sessions today to survive brief breaks
in connectivity by retaining state at end hosts before a timeout
occurs. It would be desirable to support similar session continuity
in MPTCP; however, the circumstances could be different. Whilst in
regular TCP the IP addresses will remain constant across the break in
connectivity, in MPTCP a different interface may appear. It is
desirable (but not mandated) to support this kind of "break-before-
make" session continuity. This places constraints on security
mechanisms, however, as discussed in Section 5.8. Timeouts for this
function would be locally configured.

2.2.2. Network Compatibility

In the traditional Internet architecture, network devices operate at
the network layer and lower layers, with the layers above the network
layer instantiated only at the end hosts. While this architecture,
shown in Figure 2, was initially largely adhered to, this layering no
longer reflects the "ground truth" in the Internet with the
proliferation of middleboxes [9]. Middleboxes routinely interpose on
the transport layer; sometimes even completely terminating transport
connections, thus leaving the application layer as the first real
end-to-end layer, as shown in Figure 3.

```
+-------------+                                   +-------------+
| Application |<------------ end-to-end ------------>| Application |
+-------------+                                   +-------------+
|  Transport  |<------------ end-to-end ------------>|  Transport  |
+-------------+   +-------------+   +-------------+   +-------------+
|   Network   |<->|   Network   |<->|   Network   |<->|   Network   |
+-------------+   +-------------+   +-------------+   +-------------+
   End Host            Router            Router          End Host
```
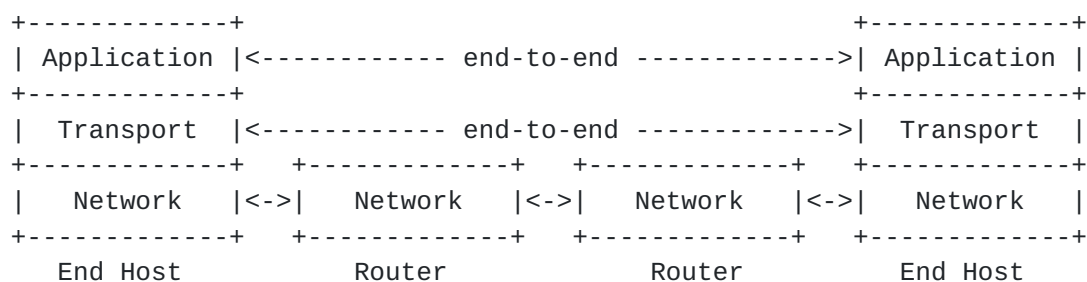
Figure 2: Traditional Internet Architecture

```
+-------------+                                      +-------------+
| Application |<------------ end-to-end ------------>| Application |
+-------------+                     +-------------+  +-------------+
|  Transport  |<------------------->|  Transport  |<->|  Transport  |
+-------------+   +-------------+    +-------------+  +-------------+
|   Network   |<->|   Network   |<->|   Network   |<->|   Network   |
+-------------+   +-------------+    +-------------+  +-------------+
                                       Firewall,
    End Host           Router        NAT, or Proxy      End Host
```
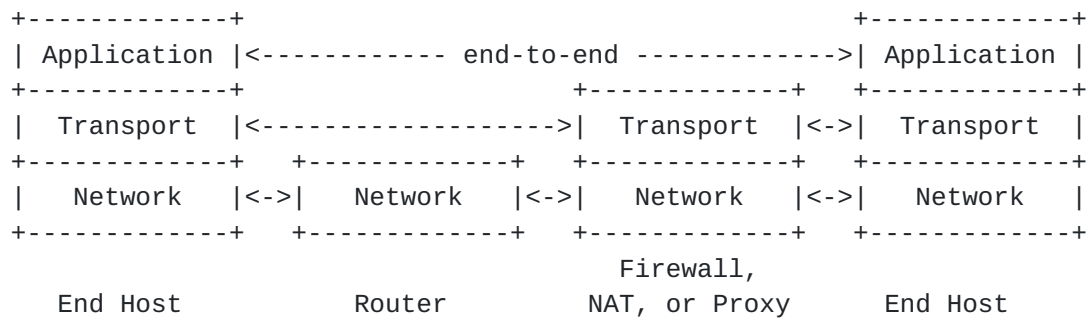
Figure 3: Internet Reality

Middleboxes that interpose on the transport layer result in loss of
"fate-sharing" [10], that is, they often hold "hard" state that, when
lost or corrupted, results in loss or corruption of the end-to-end
transport connection.

The network compatibility goal requires that the multipath extension
to TCP retain compatibility with the Internet as it exists today,
including making reasonable efforts to be able to traverse
predominant middleboxes such as firewalls, NATs, and performance-
enhancing proxies [9]. This requirement comes from recognizing
middleboxes as a significant deployment bottleneck for any transport
that is not TCP or UDP, and constrains Multipath TCP to appear as TCP
does on the wire and to use established TCP extensions where
necessary. To ensure "end-to-endness" of the transport, Multipath
TCP MUST preserve fate-sharing without making any assumptions about
middlebox behavior.

A detailed analysis of middlebox behavior and the impact on the
Multipath TCP architecture is presented in Section 7. In addition,
network compatibility must be retained to the extent that Multipath
TCP MUST fall back to regular TCP if there are insurmountable
incompatibilities for the multipath extension on a path.

Middleboxes may also cause some TCP features to be able to exist on
one subflow but not another. Typically, these will be at the subflow
level (such as selective acknowledgment (SACK) [11]) and thus do not
affect the connection-level behavior. In the future, any proposed
TCP connection-level extensions should consider how they can coexist
with MPTCP.

The modifications to support Multipath TCP remain at the transport
layer, although some knowledge of the underlying network layer is
required. Multipath TCP SHOULD work with IPv4 and IPv6
interchangeably, i.e., one connection may operate over both IPv4 and
IPv6 networks.

2.2.3. Compatibility with Other Network Users

As a corollary to both network and application compatibility, the
architecture must enable new Multipath TCP flows to coexist
gracefully with existing single-path TCP flows, competing for
bandwidth neither unduly aggressively nor unduly timidly (unless low-
precedence operation is specifically requested by the application,
such as with LEDBAT). The use of multiple paths MUST NOT unduly harm
users using single-path TCP at shared bottlenecks, beyond the impact
that would occur from another single-path TCP flow. Multiple
Multipath TCP flows on a shared bottleneck MUST share bandwidth
between each other with similar fairness to that which occurs at a
shared bottleneck with single-path TCP.

2.3. Security Goals

The extension of TCP with multipath capabilities will bring with it a
number of new threats, analyzed in detail in [12]. The security goal
for Multipath TCP is to provide a service no less secure than
regular, single-path TCP. This will be achieved through a
combination of existing TCP security mechanisms (potentially modified
to align with the Multipath TCP extensions) and of protection against
the new multipath threats identified. The design decisions derived
from this goal are presented in Section 5.8.

2.4. Related Protocols

There are several similarities between SCTP [6] and MPTCP, in that
both can make use of multiple addresses at end hosts to give some
multipath capability. In SCTP, the primary use case is to support
redundancy and mobility for multihomed hosts (i.e., a single path
will change one of its end host addresses); the simultaneous use of
multiple paths is not supported. Extensions are proposed to support
simultaneous multipath transport [13], but these are yet to be
standardized. By far the most widely used stream-based transport
protocol is, however, TCP [1], and SCTP does not meet the network and
application compatibility goals specified in Section 2.2. For
network compatibility, there are issues with various middleboxes
(especially NATs) that are unaware of SCTP and consequently end up
blocking it. For application compatibility, applications need to
actively choose to use SCTP, and with the deployment issues, very few
choose to do so. MPTCP's compatibility goals are in part based on
these observations of SCTP's deployment issues.

3. An Architectural Basis for Multipath TCP

   This section presents one possible transport architecture that the
   authors believe can effectively support the goals for Multipath TCP.
   The new Internet model described here is based on ideas proposed
   earlier in Tng ("Transport next-generation") [14]. While by no means
   the only possible architecture supporting multipath transport, Tng
   incorporates many lessons learned from previous transport research
   and development practice, and offers a strong starting point from
   which to consider the extant Internet architecture and its bearing on
   the design of any new Internet transports or transport extensions.

```
        +------------------+
        |    Application    |
        +-----------------+  ^ Application-oriented transport
        |                 |  | functions (Semantic Layer)
        + - - Transport - -+ ----------------------------------
        |                 |  | Network-oriented transport
        +-----------------+  v functions (Flow+Endpoint Layer)
        |     Network      |
        +-----------------+
          Existing Layers            Tng Decomposition
```
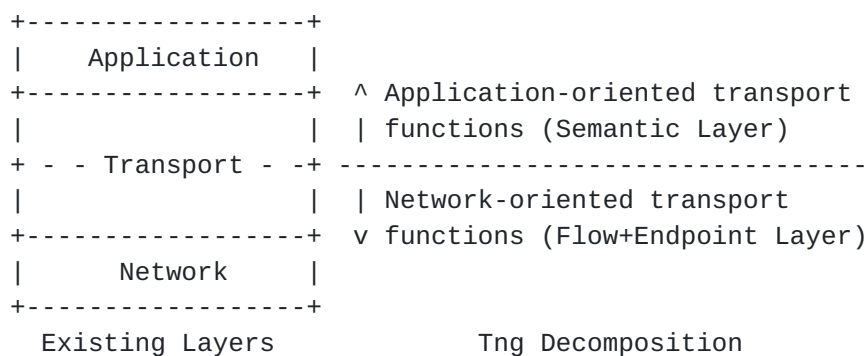
              Figure 4: Decomposition of Transport Functions

   Tng loosely splits the transport layer into "application-oriented"
   and "network-oriented" layers, as shown in Figure 4. The
   application-oriented "Semantic" layer implements functions driven
   primarily by concerns of supporting and protecting the application's
   end-to-end communication, while the network-oriented "Flow+Endpoint"
   layer implements functions such as endpoint identification (using
   port numbers) and congestion control. These network-oriented
   functions, while traditionally located in the ostensibly "end-to-end"
   Transport layer, have proven in practice to be of great concern to
   network operators and the middleboxes they deploy in the network to
   enforce network usage policies [15] [16] or optimize communication
   performance [17]. Figure 5 shows how middleboxes interact with
   different layers in this decomposed model of the transport layer: the
   application-oriented layer operates end-to-end, while the network-
   oriented layer operates "segment-by-segment" and can be interposed
   upon by middleboxes.

```
+-------------+                              +-------------+
| Application |<----------- end-to-end ------------->| Application |
+-------------+                              +-------------+
|  Semantic   |<----------- end-to-end ------------->|  Semantic   |
+-------------+  +-------------+  +-------------+  +-------------+
|Flow+Endpoint|<->|Flow+Endpoint|<->|Flow+Endpoint|<->|Flow+Endpoint|
+-------------+  +-------------+  +-------------+  +-------------+
|   Network   |<->|   Network   |<->|   Network   |<->|   Network   |
+-------------+  +-------------+  +-------------+  +-------------+
                   Firewall        Performance
    End Host       or NAT       Enhancing Proxy      End Host
```
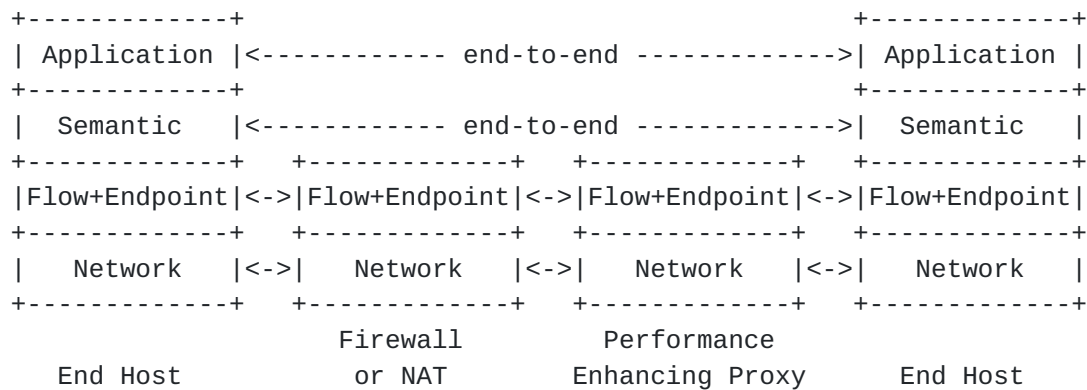
              Figure 5: Middleboxes in the New Internet Model

   MPTCP's architectural design follows Tng's decomposition as shown in
   Figure 6. MPTCP, which provides application compatibility through
   the preservation of TCP-like semantics of global ordering of
   application data and reliability, is an instantiation of the
   "application-oriented" Semantic layer; whereas the subflow TCP
   component, which provides network compatibility by appearing and
   behaving as a TCP flow in the network, is an instantiation of the
   "network-oriented" Flow+Endpoint layer.

```
   +-------------------------+    +------------------------------+
   |       Application       |    |          Application         |
   +-------------------------+    +------------------------------+
   |        Semantic         |    |            MPTCP             |
   |-----------+-------------|    + - - - - - - + - - - - - - - +
   | Flow+Endpt | Flow+Endpt |    | Subflow (TCP) | Subflow (TCP) |
   +-----------+-------------+    +-------------+---------------+
   |  Network  |   Network   |    |     IP      |      IP       |
   +-----------+-------------+    +-------------+---------------+
```
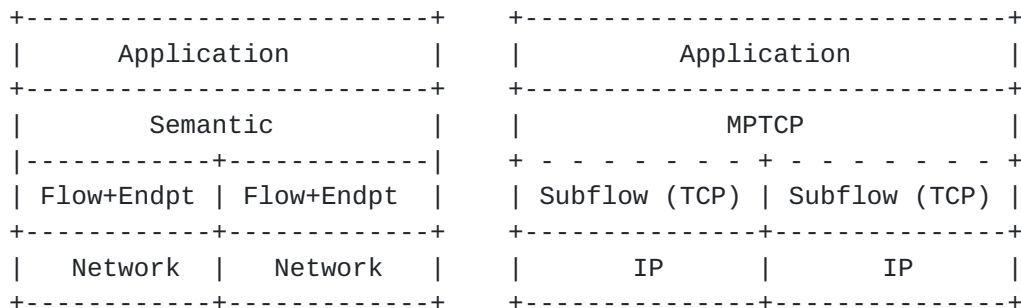
        Figure 6: Relationship between Tng (Left) and MPTCP (Right)

   As a protocol extension to TCP, MPTCP thus explicitly acknowledges
   middleboxes in its design, and specifies a protocol that operates at
   two scales: the MPTCP component operates end-to-end, while it allows
   the TCP component to operate segment-by-segment.

[4](#). A Functional Decomposition of MPTCP

   The previous two sections have discussed the goals for a Multipath
   TCP design, and provided a basis for decomposing the functions of a
   transport protocol in order to better understand the form a solution
   should take. This section builds upon this analysis by presenting
   the functional components that are used within the MPTCP design.

MPTCP makes use of (what appear to the network to be) standard TCP
sessions, termed "subflows", to provide the underlying transport per
path, and as such these retain the network compatibility desired.
MPTCP-specific information is carried in a TCP-compatible manner,
although this mechanism is separate from the actual information being
transferred so could evolve in future revisions. Figure 7
illustrates the layered architecture.

```
                                  +-------------------------------+
                                  |          Application          |
      +---------------+           +-------------------------------+
      |  Application  |           |             MPTCP             |
      +---------------+           + - - - - - - + - - - - - - - - +
      |      TCP      |           | Subflow (TCP) | Subflow (TCP) |
      +---------------+           +-------------------------------+
      |      IP       |           |      IP       |      IP       |
      +---------------+           +-------------------------------+
```
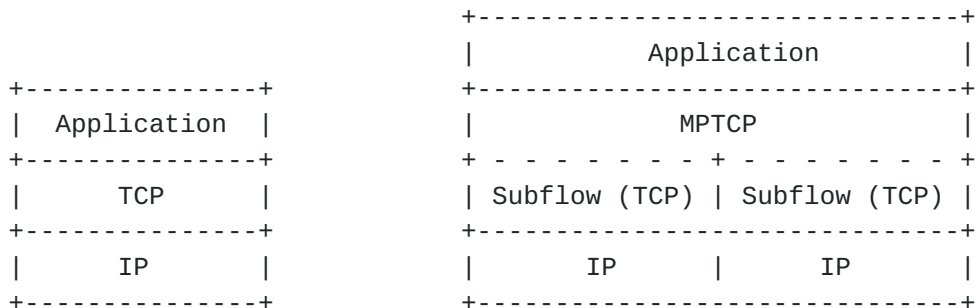
Figure 7: Comparison of Standard TCP and MPTCP Protocol Stacks

Situated below the application, the MPTCP extension in turn manages
multiple TCP subflows below it. In order to do this, it must
implement the following functions:

o Path Management: This is the function to detect and use multiple
   paths between two hosts. MPTCP uses the presence of multiple IP
   addresses at one or both of the hosts as an indicator of this.
   The path management features of the MPTCP protocol are the
   mechanisms to signal alternative addresses to hosts, and
   mechanisms to set up new subflows joined to an existing MPTCP
   connection.

o Packet Scheduling: This function breaks the byte stream received
   from the application into segments to be transmitted on one of the
   available subflows. The MPTCP design makes use of a data sequence
   mapping, associating segments sent on different subflows to a
   connection-level sequence numbering, thus allowing segments sent
   on different subflows to be correctly re-ordered at the receiver.
   The packet scheduler is dependent upon information about the
   availability of paths exposed by the path management component,
   and then makes use of the subflows to transmit queued segments.
   This function is also responsible for connection-level re-ordering
   on receipt of packets from the TCP subflows, according to the
   attached data sequence mappings.

o Subflow (single-path TCP) Interface: A subflow component takes
   segments from the packet-scheduling component and transmits them
   over the specified path, ensuring detectable delivery to the host.

      MPTCP uses TCP underneath for network compatibility; TCP ensures
      in-order, reliable delivery. TCP adds its own sequence numbers to
      the segments; these are used to detect and retransmit lost packets
      at the subflow layer. On receipt, the subflow passes its
      reassembled data to the packet scheduling component for
      connection-level reassembly; the data sequence mapping from the
      sender's packet scheduling component allows re-ordering of the
      entire byte stream.

   o Congestion Control: This function coordinates congestion control
      across the subflows. As specified, this congestion control
      algorithm MUST ensure that an MPTCP connection does not unfairly
      take more bandwidth than a single path TCP flow would take at a
      shared bottleneck. An algorithm to support this is specified in
      [7].

   These functions fit together as follows. The path management looks
   after the discovery (and if necessary, initialization) of multiple
   paths between two hosts. The packet scheduler then receives a stream
   of data from the application destined for the network, and undertakes
   the necessary operations on it (such as segmenting the data into
   connection-level segments, and adding a connection-level sequence
   number) before sending it on to a subflow. The subflow then adds its
   own sequence number, ACKs, and passes them to network. The receiving
   subflow re-orders data (if necessary) and passes it to the packet
   scheduling component, which performs connection level re-ordering,
   and sends the data stream to the application. Finally, the
   congestion control component exists as part of the packet scheduling,
   in order to schedule which segments should be sent at what rate on
   which subflow.

5. High-Level Design Decisions

   There is seemingly a wide range of choices when designing a multipath
   extension to TCP. However, the goals as discussed earlier in this
   document constrain the possible solutions, leaving relative little
   choice in many areas. This section outlines high-level design
   choices that draw from the architectural basis discussed earlier in
   Section 3, which the design of MPTCP [5] takes into account.

5.1. Sequence Numbering

   MPTCP uses two levels of sequence spaces: a connection-level sequence
   number and another sequence number for each subflow. This permits
   connection-level segmentation and reassembly and retransmission of
   the same part of connection-level sequence space on different
   subflow-level sequence space.

The alternative approach would be to use a single connection-level
sequence number, which gets sent on multiple subflows. This has two
problems: first, the individual subflows will appear to the network
as TCP sessions with gaps in the sequence space; this in turn may
upset certain middleboxes such as intrusion detection systems, or
certain transparent proxies, and would thus go against the network
compatibility goal. Second, the sender would not be able to
attribute packet losses or receptions to the correct path when the
same segment is sent on multiple paths (i.e., in the case of
retransmissions).

The sender must be able to tell the receiver how to reassemble the
data, for delivery to the application. In order to achieve this, the
receiver must determine how subflow-level data (carrying subflow
sequence numbers) maps at the connection level. This is referred to
as the "data sequence mapping". This mapping can be represented as a
tuple of (data sequence number, subflow sequence number, length),
i.e., for a given number of bytes (the length), the subflow sequence
space beginning at the given sequence number maps to the connection-
level sequence space (beginning at the given data sequence number).
This information could conceivably have various sources.

One option to signal the data sequence mapping would be to use
existing fields in the TCP segment (such as subflow sequence number,
length) and add only the data sequence number to each segment, for
instance, as a TCP option. This would be vulnerable, however, to
middleboxes that re-segment or assemble data, since there is no
specified behavior for coalescing TCP options. If one signaled (data
sequence number, length), this would still be vulnerable to
middleboxes that coalesce segments and do not understand MPTCP
signaling so do not correctly rewrite the options.

Because of these potential issues, the design decision taken in the
MPTCP protocol is that whenever a mapping for subflow data needs to
be conveyed to the other host, all three pieces of data (data seq,
subflow seq, length) must be sent. To reduce the overhead, it would
be permissible for the mapping to be sent periodically and cover more
than a single segment. Further experimentation is required to
determine what tradeoffs exist regarding the frequency at which
mappings should be sent. It could also be excluded entirely in the
case of a connection before more than one subflow is used, where the
data-level and subflow-level sequence space is the same.

5.2. Reliability and Retransmissions

MPTCP features acknowledgements at connection-level as well as
subflow-level acknowledgements, in order to provide a robust service
to the application.

   Under normal behavior, MPTCP could use the data sequence mapping and
   subflow ACKs to decide when a connection-level segment was received.
   The transmission of TCP ACKs for a subflow are handled entirely at
   the subflow level, in order to maintain TCP semantics and trigger
   subflow-level retransmissions. This has certain implications on end-
   to-end semantics. It would mean that once a segment is ACKed at the
   subflow level, it cannot be discarded in the re-order buffer at the
   connection level. Secondly, unlike in standard TCP, a receiver
   cannot simply drop out-of-order segments if needed (for instance, due
   to memory pressure). Under certain circumstances, it may be
   desirable to drop segments after acknowledgement on the subflow but
   before delivery to the application, and this can be facilitated by a
   connection-level acknowledgement.

   Furthermore, it is possible to conceive of some cases where
   connection-level acknowledgements could improve robustness. Consider
   a subflow traversing a transparent proxy: if the proxy ACKs a segment
   and then crashes, the sender will not retransmit the lost segment on
   another subflow, as it thinks the segment has been received. The
   connection grinds to a halt despite having other working subflows,
   and the sender would be unable to determine the cause of the problem.
   An example situation where this may occur would be mobility between
   wireless access points, each of which operates a transport-level
   proxy. Finally, as an optimization, it may be feasible for a
   connection-level acknowledgement to be transmitted over the shortest
   Round-Trip Time (RTT) path, potentially reducing send buffer
   requirements (see Section 5.3).

   Therefore, to provide a fully robust multipath TCP solution given the
   above constraints, MPTCP for use on the public Internet MUST feature
   explicit connection-level acknowledgements, in addition to subflow-
   level acknowledgements. A connection-level acknowledgement would
   only be required in order to signal when the receive window moves
   forward; the heuristics for using such a signal are discussed in more
   detail in the protocol specification [5].

   Regarding retransmissions, it MUST be possible for a segment to be
   retransmitted on a different subflow from that on which it was
   originally sent. This is one of MPTCP's core goals, in order to
   maintain integrity during temporary or permanent subflow failure, and
   this is enabled by the dual sequence number space.

   The scheduling of retransmissions will have significant impact on
   MPTCP user experience. The current MPTCP specification suggests that
   data outstanding on subflows that have timed out should be
   rescheduled for transmission on different subflows. This behavior

aims to minimize disruption when a path breaks, and uses the first
timeout as indicators. More conservative versions would be to use
second or third timeouts for the same segment.

Typically, fast retransmit on an individual subflow will not trigger
retransmission on another subflow, although this may still be
desirable in certain cases, for instance, to reduce the receive
buffer requirements. However, in all cases with retransmissions on
different subflows, the lost segments SHOULD still be sent on the
path that lost them. This is currently believed to be necessary to
maintain subflow integrity, as per the network compatibility goal.
By doing this, some efficiency is lost, and it is unclear at this
point what the optimal retransmit strategy is.

Large-scale experiments are therefore required in order to determine
the most appropriate retransmission strategy, and recommendations
will be refined once more information is available.

5.3. Buffers

To ensure in-order delivery, MPTCP must use a connection level
receive buffer, where segments are placed until they are in order and
can be read by the application.

In regular, single-path TCP, it is usually recommended to set the
receive buffer to 2*BDP (Bandwidth-Delay Product, i.e., BDP = BW*RTT,
where BW = Bandwidth and RTT = Round-Trip Time). One BDP allows
supporting reordering of segments by the network. The other BDP
allows the connection to continue during fast retransmit: when a
segment is fast retransmitted, the receiver must be able to store
incoming data during one more RTT.

For MPTCP, the story is a bit more complicated. The ultimate goal is
that a subflow packet loss or subflow failure should not affect the
throughput of other working subflows; the receiver should have enough
buffering to store all data until the missing segment is re-
transmitted and reaches the destination.

The worst-case scenario would be when the subflow with the highest
RTT/RTO (Round-Trip Time or Retransmission TimeOut) experiences a
timeout; in that case, the receiver has to buffer data from all
subflows for the duration of the RTO. Thus, the smallest connection-
level receive buffer that would be needed to avoid stalling with
subflow failures is sum(BW_i)*RTO_max, where BW_i = Bandwidth for
each subflow and RTO_max is the largest RTO across all subflows.

This is an order of magnitude more than the receive buffer required
for a single connection, and is probably too expensive for practical
purposes. A more sensible requirement is to avoid stalls in the
absence of timeouts. Therefore, the RECOMMENDED receive buffer is
2*sum(BW_i)*RTT_max, where RTT_max is the largest RTT across all
subflows. This buffer sizing ensures subflows do not stall when fast
retransmit is triggered on any subflow.

The resulting buffer size should be small enough for practical use.
However, there may be extreme cases where fast, high throughput paths
(e.g., 100 Mb/s, 10 ms RTT) are used in conjunction with slow paths
(e.g., 1 Mb/s, 1000 ms RTT). In that case, the required receive
buffer would be 12.5 MB, which is likely too big. In extreme cases
such as this example, it may be prudent to only use some of the
fastest available paths for the MPTCP connection, potentially using
the slow path(s) for backup only.

Send Buffer: The RECOMMENDED send buffer is the same size as the
recommended receive buffer, i.e., 2*sum(BW_i)*RTT_max. This is
because the sender must locally store the segments sent but
unacknowledged by the connection level ACK. The send buffer size
matters particularly for hosts that maintain a large number of
ongoing connections. If the required send buffer is too large, a
host can choose to only send data on the fast subflows, using the
slow subflows only in cases of failure.

5.4. Signaling

Since MPTCP uses TCP as its subflow transport mechanism, an MPTCP
connection will also begin as a single TCP connection. Nevertheless,
it must signal to the peer that it supports MPTCP and wishes to use
it on this connection. As such, a TCP option will be used to
transmit this information, since this is the established mechanism
for indicating additional functionality on a TCP session.

In addition, further signaling is required during the operation of an
MPTCP session, such as that for reassembly across multiple subflows,
and for informing the other host about other available IP addresses.

The MPTCP protocol design will use TCP options for this additional
signaling. This has been chosen as the mechanism most fitting in
with the goals as specified in Section 2. With this mechanism, the
signaling required to operate MPTCP is transported separately from
the data, allowing it to be created and processed separately from the
data stream, and retaining architectural compatibility with network
entities.

This decision is the consensus of the Working Group (following
detailed discussions at IETF78), and the main reasons for this are as
follows:

o TCP options are the traditional signaling method for TCP;

o A TCP option on a SYN is the most compatible way for an end host
   to signal it is MPTCP capable;

o If connection-level ACKs are signaled in the payload, then they
   may suffer from packet loss and may be congestion-controlled,
   which may affect the data throughput in the forward direction and
   could lead to head-of-line blocking;

o Middleboxes, such as NAT traversal helpers, can easily parse TCP
   options, e.g., to rewrite addresses.

On the other hand, the main drawbacks of TCP options compared to TLV
encoding in the payload are the following:

o There is limited space for signaling messages;

o A middlebox may, potentially, drop a packet with an unknown
   option;

o The transport of control information in options is not necessarily
   reliable.

The detailed design of MPTCP alleviates these issues as far as
possible by carefully considering the size of MPTCP options and
seamlessly falling back to regular TCP on the loss of control data.

Both option and payload encoding may interfere with offloading of TCP
processing to high-speed network interface cards, such as
segmentation, checksumming, and reassembly. For network cards
supporting MPTCP, signaling in TCP options should simplify offloading
due to the separate handling of MPTCP signaling and data.

5.5. Path Management

Currently, the network does not expose path diversity between pairs
of IP addresses. In order to achieve path diversity from today's IP
networks, in the typical case, MPTCP uses multiple addresses at one
or both hosts to infer different paths across the network. It is
expected that these paths, whilst not necessarily entirely non-
overlapping, will be sufficiently disjoint to allow multipath to

achieve improved throughput and robustness. The use of multiple IP
addresses is a simple mechanism that requires no additional features
in the network.

Multiple different (source, destination) address pairs will thus be
used as path selectors in most cases. However, each path will be
identified by a standard five-tuple (i.e., source address,
destination address, source port, destination port, protocol), which
can allow the extension of MPTCP to use ports as well as addresses as
path selectors. This will allow hosts to use port-based load
balancing with MPTCP, for example, if the network routes different
ports over different paths (which may be the case with technologies
such as Equal Cost MultiPath (ECMP) routing [4]). It should be
noted, however, that ISPs often undertake traffic engineering in
order to optimize resource utilization within their networks, and
care should be taken (by both ISPs and developers) that MPTCP using
broadly similar paths does not adversely interfere with this.

For an increased chance of successfully setting up additional
subflows (such as when one end is behind a firewall, NAT, or other
restrictive middlebox), either host SHOULD be able to add new
subflows to an MPTCP connection. MPTCP MUST be able to handle paths
that appear and disappear during the lifetime of a connection (for
example, through the activation of an additional network interface).

The path management is a separate function from the packet
scheduling, subflow interface, and congestion control functions of
MPTCP, as documented in Section 4. As such, it would be feasible to
replace this IP-address-based design with an alternative path
selection mechanism in the future, with no significant changes to the
other functional components.

5.6. Connection Identification

Since an MPTCP connection may not be bound to a traditional 5-tuple
(source address and port, destination address and port, protocol
number) for the entirety of its existence, it is desirable to provide
a new mechanism for connection identification. This will be useful
for MPTCP-aware applications and for the MPTCP implementation (and
MPTCP-aware middleboxes) to have a unique identifier with which to
associate the multiple subflows.

Therefore, each MPTCP connection requires a connection identifier at
each host, which is locally unique within that host. In many ways,
this is analogous to an ephemeral port number in regular TCP. The
manifestation and purpose of such an identifier is out of the scope
of this architecture document.

   Non-MPTCP-aware applications will not, however, have access to this
   identifier and in such cases an MPTCP connection will be identified
   by the 5-tuple of the first TCP subflow. It is out of the scope of
   this document, however, to define the behavior of the MPTCP
   implementation if the first TCP subflow later fails. If there are
   MPTCP-unaware applications that make assumptions about continued
   existence of the initial address pair, their behavior could be
   disrupted by carrying on regardless. It is expected that this is a
   very small, possibly negligible, set of applications, however. MPTCP
   MUST NOT be used for applications that request to bind to a specific
   address or interface, since such applications are making a deliberate
   choice of path in use.

   Since the requirements of applications are not clear at this stage,
   however, it is as yet unconfirmed whether carrying on in the event of
   the loss of the initial address pair would be a damaging assumption
   to make. This behavior will be an implementation-specific solution,
   and as such it is expected to be chosen by implementors once more
   research has been undertaken to determine its impact.

5.7. Congestion Control

   As discussed in network-layer compatibility requirements
   Section 2.2.3, there are three goals for the congestion control
   algorithms used by an MPTCP implementation: improve throughput (at
   least as well as a single-path TCP connection would perform); do no
   harm to other network users (do not take up more capacity on any one
   path than if it was a single path flow using only that route -- this
   is particularly relevant for shared bottlenecks); and balance
   congestion by moving traffic away from the most congested paths. To
   achieve these goals, the congestion control algorithms on each
   subflow must be coupled in some way. A proposal for a suitable
   congestion control algorithm is given in [7].

5.8. Security

   A detailed threat analysis for Multipath TCP is presented in a
   separate document [12]. That document focuses on flooding attacks
   and hijacking attacks that can be launched against a Multipath TCP
   connection.

   The basic security goal of Multipath TCP, as introduced in
   Section 2.3, can be stated as: "provide a solution that is no worse
   than standard TCP".

From the threat analysis, and with this goal in mind, three key
security requirements can be identified. A multi-addressed Multipath
TCP SHOULD be able to do the following:

o Provide a mechanism to confirm that the parties in a subflow
   handshake are the same as in the original connection setup (e.g.,
   require use of a key exchanged in the initial handshake in the
   subflow handshake, to limit the scope for hijacking attacks).

o Provide verification that the peer can receive traffic at a new
   address before adding it (i.e., verify that the address belongs to
   the other host, to prevent flooding attacks).

o Provide replay protection, i.e., ensure that a request to add/
   remove a subflow is 'fresh'.

Additional mechanisms have been deployed as part of standard TCP
stacks to provide resistance to Denial-of-Service (DoS) attacks. For
example, there are various mechanisms to protect against TCP reset
attacks [18], and Multipath TCP should continue to support similar
protection. In addition, TCP SYN Cookies [19] were developed to
allow a TCP server to defer the creation of session state in the
SYN_RCVD state, and remain stateless until the ESTABLISHED state had
been reached. Multipath TCP should, ideally, continue to provide
such functionality and, at a minimum, avoid significant computational
burden prior to reaching the ESTABLISHED state (of the Multipath TCP
connection as a whole).

It should be noted that aspects of the Multipath TCP design space
place constraints on the security solution:

o The use of TCP options significantly limits the amount of
   information that can be carried in the handshake.

o The need to work through middleboxes results in the need to handle
   mutability of packets.

o The desire to support a 'break-before-make' (as well as a 'make-
   before-break') approach to adding subflows (within a limited time
   period) implies that a host cannot rely on using a pre-existing
   subflow to support the addition of a new one.

The MPTCP protocol will be designed with these security requirements
in mind, and the protocol specification [5] will document how these
are met.

[6](#). Software Interactions

[6.1](#). Interactions with Applications

   In the case of applications that have used an existing API call to
   bind to a specific address or interface, the MPTCP extension MUST NOT
   be used. This is because the applications are indicating a clear
   choice of path to use and thus will have expectations of behavior
   that must be maintained, in order to adhere to the application
   compatibility goals.

   Interactions with applications are presented in [[8](#)] -- including, but
   not limited to, performances changes that may be expected, semantic
   changes, and new features that may be requested through an enhanced
   API.

   TCP features the ability to send "Urgent" data, the delivery of which
   to the application may or may not be out-of-band. The use of this
   feature is not recommended due to security implications and
   implementation differences [[20](#)]. MPTCP requires contiguous data to
   support its data sequence mapping over multiple segments, and
   therefore the Urgent pointer cannot interrupt an existing mapping.
   An MPTCP implementation MAY choose to support sending Urgent data,
   and if it does, it SHOULD send the Urgent data on the soonest
   available unassigned subflow sequence space. Incoming Urgent data
   SHOULD be mapped to connection-level sequence space and delivered to
   the application analogous to Urgent data in regular TCP.

[6.2](#). Interactions with Management Systems

   To enable interactions between TCP and network management systems,
   the TCP [[21](#)] and TCP Extended Statistics (ESTATS) [[22](#)] MIBs have been
   defined. MPTCP should share these MIBs for aspects that are designed
   to be transparent to the application.

   It is anticipated that an MPTCP MIB will be defined in the future,
   once experience of experimental MPTCP deployments is gathered. This
   MIB would provide access to MPTCP-specific properties such as whether
   MPTCP is enabled and the number and properties of the individual
   paths in use.

[7](#). Interactions with Middleboxes

   As discussed in [Section 2.2](#), it is a goal of MPTCP to be deployable
   today and thus compatible with the majority of middleboxes. This
   section summarizes the issues that may arise with NATs, firewalls,
   proxies, intrusion detection systems, and other middleboxes that, if
   not considered in the protocol design, may hinder its deployment.

This section is intended primarily as a description of options and
considerations only. Protocol-specific solutions to these issues
will be given in the companion documents.

Multipath TCP will be deployed in a network that no longer provides
just basic datagram delivery. A myriad of middleboxes are deployed
to optimize various perceived problems with the Internet protocols:
NATs primarily address IP address space shortage [15], Performance
Enhancing Proxies (PEPs) optimize TCP for different link
characteristics [17], firewalls [16] and intrusion detection systems
try to block malicious content from reaching a host, and traffic
normalizers [23] ensure a consistent view of the traffic stream to
Intrusion Detection Systems (IDS) and hosts.

All these middleboxes optimize current applications at the expense of
future applications. In effect, future applications will often need
to behave in a similar fashion to existing ones, in order to increase
the chances of successful deployment. Further, the precise behavior
of all these middleboxes is not clearly specified, and implementation
errors make matters worse, raising the bar for the deployment of new
technologies.

The following list of middlebox classes documents behavior that could
impact the use of MPTCP. This list is used in [5] to describe the
features of the MPTCP protocol that are used to mitigate the impact
of these middlebox behaviors.

o NATs: Network Address Translators decouple the host's local IP
   address (and, in the case of NAPTs, port) with that which is seen
   in the wider Internet when the packets are transmitted through a
   NAT. This adds complexity, and reduces the chances of success,
   when signaling IP addresses.

o PEPs: Performance Enhancing Proxies, which aim to improve the
   performance of protocols over low-performance (e.g., high-latency
   or high-error-rate) links. As such, they may "split" a TCP
   connection and behavior such as proactive ACKing may occur, and
   therefore it is no longer guaranteed that one host is
   communicating directly with another. PEPs, firewalls, or other
   middleboxes may also change the declared receive window size.

o Traffic Normalizers: These aim to eliminate ambiguities and
   potential attacks at the network level, and amongst other things,
   are unlikely to permit holes in TCP-level sequence space (which
   has an impact on MPTCP's retransmission and subflow sequence
   numbering design choices).

o Firewalls: on top of preventing incoming connections, firewalls
   may also attempt additional protection such as sequence number
   randomization (so a sender cannot reliably know what TCP sequence
   number the receiver will see).

o IDSs: Intrusion Detection Systems may look for traffic patterns to
   protect a network and may have false positives with MPTCP and drop
   the connections during normal operation. Future MPTCP-aware
   middleboxes will require the ability to correlate the various
   paths in use.

o Content-Aware Firewalls: Some middleboxes may actively change data
   in packets, such as rewriting URIs in HTTP traffic.

In addition, all classes of middleboxes may affect TCP traffic in the
following ways:

o TCP Options: some middleboxes may drop packets with unknown TCP
   options or strip those options from the packets.

o Segmentation and Coalescing: middleboxes (or even something as
   close to the end host as TCP Segmentation Offloading (TSO) on a
   Network Interface Card (NIC)) may change the packet boundaries
   from those that the sender intended. It may do this by splitting
   packets or coalescing them together. This leads to two major
   impacts: where a packet boundary will be cannot be guaranteed and
   what a middlebox will do with TCP options in these cases (they may
   be repeated, dropped, or sent only once) cannot be said for sure.

[8](#). Contributors

The authors would like to acknowledge the contributions of Andrew
McDonald and Bryan Ford to this document.

The authors would also like to thank the following people for
detailed reviews: Olivier Bonaventure, Gorry Fairhurst, Iljitsch van
Beijnum, Philip Eardley, Michael Scharf, Lars Eggert, Cullen
Jennings, Joel Halpern, Juergen Quittek, Alexey Melnikov, David
Harrington, Jari Arkko, and Stewart Bryant.

[9](#). Acknowledgements

10. Security Considerations

   This informational document provides an architectural overview for
   Multipath TCP and so does not, in itself, raise any security issues.
   A separate threat analysis [12] lists threats that can exist with a
   Multipath TCP. However, a protocol based on the architecture in this
   document will have a number of security requirements. The high-level
   goals for such a protocol are identified in Section 2.3, whilst
   Section 5.8 provides more detailed discussion of security
   requirements and design decisions which are applied in the MPTCP
   protocol design [5].

11. References

11.1. Normative References

   [1]   Postel, J., "Transmission Control Protocol", STD 7, RFC 793,
          September 1981.

   [2]   Bradner, S., "Key words for use in RFCs to Indicate Requirement
          Levels", BCP 14, RFC 2119, March 1997.

11.2. Informative References

   [3]   Wischik, D., Handley, M., and M. Bagnulo Braun, "The Resource
          Pooling Principle", ACM SIGCOMM CCR vol. 38 num. 5, pp. 47-52,
          October 2008,
          <http://ccr.sigcomm.org/online/files/p47-handleyA4.pdf>.

   [4]   Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm",
          RFC 2992, November 2000.

   [5]   Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP
          Extensions for Multipath Operation with Multiple Addresses",
          Work in Progress, March 2011.

   [6]   Stewart, R., "Stream Control Transmission Protocol", RFC 4960,
          September 2007.

   [7]   Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion
          Control for Multipath Transport Protocols", Work in Progress,
          March 2011.

   [8]   Scharf, M. and A. Ford, "MPTCP Application Interface
          Considerations", Work in Progress, March 2011.

   [9]   Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues",
          RFC 3234, February 2002.

[10]   Carpenter, B., "Internet Transparency", RFC 2775,
         February 2000.

[11]   Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP
         Selective Acknowledgment Options", RFC 2018, October 1996.

[12]   Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath
         Operation with Multiple Addresses", RFC 6181, March 2011.

[13]   Becke, M., Dreibholz, T., Iyengar, J., Natarajan, P., and M.
         Tuexen, "Load Sharing for the Stream Control Transmission
         Protocol (SCTP)", Work in Progress, December 2010.

[14]   Ford, B. and J. Iyengar, "Breaking Up the Transport Logjam",
          ACM HotNets, October 2008.

[15]   Srisuresh, P. and K. Egevang, "Traditional IP Network Address
         Translator (Traditional NAT)", RFC 3022, January 2001.

[16]   Freed, N., "Behavior of and Requirements for Internet
         Firewalls", RFC 2979, October 2000.

[17]   Border, J., Kojo, M., Griner, J., Montenegro, G., and Z.
         Shelby, "Performance Enhancing Proxies Intended to Mitigate
         Link-Related Degradations", RFC 3135, June 2001.

[18]   Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's
         Robustness to Blind In-Window Attacks", RFC 5961, August 2010.

[19]   Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations",
         RFC 4987, August 2007.

[20]   Gont, F. and A. Yourtchenko, "On the Implementation of the TCP
         Urgent Mechanism", RFC 6093, January 2011.

[21]   Raghunarayan, R., "Management Information Base for the
         Transmission Control Protocol (TCP)", RFC 4022, March 2005.

[22]   Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended
         Statistics MIB", RFC 4898, May 2007.

[23]   Handley, M., Paxson, V., and C. Kreibich, "Network Intrusion
         Detection: Evasion, Traffic Normalization, and End-to-End
         Protocol Semantics", Usenix Security 2001, 2001, <http://
         www.usenix.org/events/sec01/full_papers/handley/handley.pdf>.

Authors' Addresses

   Alan Ford
   Roke Manor Research
   Old Salisbury Lane
   Romsey, Hampshire SO51 0ZN
   UK
   Phone: +44 1794 833 465
   EMail: alan.ford@roke.co.uk

   Costin Raiciu
   University College London
   Gower Street
   London WC1E 6BT
   UK
   EMail: c.raiciu@cs.ucl.ac.uk

   Mark Handley
   University College London
   Gower Street
   London WC1E 6BT
   UK
   EMail: m.handley@cs.ucl.ac.uk

   Sebastien Barre
   Universite catholique de Louvain
   Pl. Ste Barbe, 2
   Louvain-la-Neuve 1348
   Belgium
   Phone: +32 10 47 91 03
   EMail: sebastien.barre@uclouvain.be

   Janardhan Iyengar
   Franklin and Marshall College
   Mathematics and Computer Science
   PO Box 3003
   Lancaster, PA 17604-3003
   USA
   Phone: 717-358-4774
   EMail: jiyengar@fandm.edu