MPTCP Application Interface Considerations
draft-scharf-mptcp-api-00

Status of This Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 18, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

Multipath TCP (MPTCP) adds the capability of using multiple paths to a regular TCP session. Even though it is designed to be totally
backward compatible, the data transport differs to the existing TCP, and there are several additional degrees of freedom that affect applications. This document summarizes the impact that MPTCP may have on applications, such as changes in performance. Furthermore, it describes an optional extended application interface that provides access to multipath information and enables control of some aspects of the MPTCP implementation’s behaviour.

Table of Contents

1. Introduction .............................................. 3
2. Terminology ............................................. 3
3. Impact of MPTCP on Applications ......................... 4
   3.1. Performance Improvement ............................. 4
       3.1.1. Throughput .................................... 4
       3.1.2. Delay .......................................... 4
       3.1.3. Resilience .................................... 5
   3.2. Potential Problems .................................. 5
       3.2.1. Impact of Middleboxes ........................ 5
       3.2.2. Outdated Implicit Assumptions ................. 5
4. Implications of MPTCP on Existing Interfaces ........... 6
   4.1. Overview of the Network Stack ...................... 6
   4.2. Impact on the Use of Socket Options ............... 6
   4.3. Impact on Existing Other System-wide Settings .... 7
   4.4. Impact on Existing API Calls ...................... 8
   4.5. Impact on Existing Sockets API Enhancements ....... 8
5. Application Requirements ............................... 8
   5.1. MPTCP Usage Scenarios .............................. 8
   5.2. Requirements on API Extensions .................... 10
6. Specification of API Extensions for MPTCP .............. 11
   6.1. Design Considerations ............................. 11
   6.2. Overview of Sockets Interface Extensions .......... 12
   6.3. Detailed Description ............................... 12
       6.3.1. TCP_MP_ENABLE .................................. 12
       6.3.2. TCP_MP_MAXSUBFLOW ............................ 12
   6.4. Usage examples .................................... 12
   6.5. Discussion of Interactions ......................... 12
   6.6. Advice to Application Developers .................. 12
7. Security Considerations ............................... 13
8. IANA Considerations .................................... 13
9. Conclusion .............................................. 13
10. Acknowledgments ....................................... 13
11. References ............................................ 13
    11.1. Normative References ............................ 13
    11.2. Informative References ........................... 14
1. Introduction

Multipath TCP (MPTCP) [4] adds the capability of using multiple paths to a regular TCP session [1]. MPTCP offers the same reliable, in-order, byte-stream transport like TCP and is designed to be backward-compatible. It requires support inside the network stack of both endpoints. This document presents the impacts that MPTCP may have on applications, such as performance changes. Furthermore, it specifies an extended Application Programming Interface (API) describing how applications can exploit additional features of multipath transport. While MPTCP needs to be usable without any application changes, this API is an optional extension that provides access to multipath information and enables control of some aspects of the MPTCP implementation’s behavior.

The de facto standard API for TCP/IP applications is the "sockets" interface. This document defines experimental MPTCP-specific extensions, in particular additional socket options. It is up to the applications, or high-level programming languages or libraries, to decide whether to use these optional extensions. For instance, an application may want to turn on or off the MPTCP mechanism for certain data transfers, or provide some guidance concerning its usage. The syntax and semantics of the specification is in line with the Posix standard [5] as much as possible.

There are various related extensions of the sockets interface: [7] specifies sockets API extensions for the multihoming shim layer. The API enables interactions between applications and the multihoming shim layer for advanced locator management, and access to information about failure detection and path exploration. Other experimental extensions to the sockets API are defined for the Host Identity Protocol (HIP) [8] in order to manage the bindings of identifiers and locator. There can be interactions of these APIs with MPTCP. Other related API extensions exist for IPv6 [6]. The MPTCP API also has some similarity to the SCTP socket API [9].

The target readers of this document are application programmers who develop application software that may benefit significantly from MPTCP. This document also provides the necessary information for developers of MPTCP to implement the API in a TCP/IP network stack.

2. Terminology

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 [3].

This document uses the terminology introduced in [4].
3. Impact of MPTCP on Applications

3.1. Performance Improvement

One of the key goals of adding multipath capability to TCP is to improve the performance of a transport connection. Furthermore, it is an explicit goal of MPTCP that it should not provide a worse performing connection that would have existed through the use of legacy, single-path TCP.

3.1.1. Throughput

The most obvious performance improvement that will be gained with the use of MPTCP is an increase in throughput, since MPTCP will pool more than one path (where available) between two endpoints. This will provide greater bandwidth for an application. If there are shared bottlenecks between the flows, then the congestion control algorithms will ensure that load is correctly spread and the end user receives no worse performance than single-path TCP.

Furthermore, this means that an MPTCP session could achieve throughput that is greater than the capacity of a single interface on the device. If any applications make assumptions about interfaces due to throughput (or vice versa), they must take this into account.

A small overhead will be present, through the use of MPTCP options, and as such the impact of this when there are multiple subflows over a shared bottleneck (or bottlenecks) should be considered, but is FFS, and will be part of the definition of a suitable congestion control algorithm.

3.1.2. Delay

If the delays on the constituent subflows of an MPTCP connection differ, the jitter perceivable to an application may appear higher as the data is striped across the subflows. Although MPTCP will ensure in-order delivery to the application, the application must be able to cope with the data being burstier than may be usual with single-path TCP. Since burstiness is commonplace on the Internet today, it is unlikely that applications will suffer from such an impact on traffic profile, but application authors may wish to consider this in future development.

In addition, applications that make round trip time (RTT) estimates at the application level may have some issues. Whilst the average delay calculated will be accurate, whether this is useful for an application will depend on what it requires this information for. If a new application wishes to derive such information, it should...
consider how multiple subflows may affect its measurements, and thus how it may wish to respond. In such a case, an application may wish to express its scheduling preferences, as described later in this document.

3.1.3. Resilience

The use of multiple subflows simultaneously means that, if one should fail, all traffic will move to the remaining subflow(s), and additionally any lost packets can be retransmitted on these subflows.

Subflow failure may be caused by issues within the network, which an application would be unaware of, or interface failure on the node. An application may, under certain circumstances, be in a position to be aware of such failure (e.g. by radio signal strength, or simply an interface enabled flag), and so must not make assumptions of an MPTCP flow’s stability based on this. MPTCP will never override an application’s request for a given interface, however, so the cases where this issue may be applicable are limited.

3.2. Potential Problems

3.2.1. Impact of Middleboxes

MPTCP has been designed in order to pass through the majority of middleboxes, for example through its ability to open subflows in either direction, and through its use of a data-level sequence number.

Nevertheless some middleboxes may still refuse to pass MPTCP messages due to the presence of TCP options. If this is the case, MPTCP should fall back to regular TCP. Although this will not create a problem for the application (its communication will be set up either way), there may be additional (and indeed, user-perceivable) delay while the first handshake fails.

Empirical evidence suggests that new TCP options can successfully be used on most paths in the Internet. But they can also have other unexpected implications. For instance, intrusion detection systems could be triggered.

3.2.2. Outdated Implicit Assumptions

MPTCP overcomes the one-to-one mapping of the socket interface to a flow through the network. As a result, applications cannot implicitly rely on this one-to-one mapping any more. Applications that require the transport along a single path can disable the use of MPTCP as described later in this document. One example are
monitoring tools that want to measure the available bandwidth on a path.

Security implications: TODO

4. Implications of MPTCP on Existing Interfaces

4.1. Overview of the Network Stack

MPTCP is an extension of TCP. TCP interacts with other parts of the network stack by different interfaces. The de facto standard API between TCP and applications is the sockets interface. The position of MPTCP in the protocol stack can be illustrated as follows:

```
+-------------------------------+
|           Application         |
+-------------------------------+

^                 |
~~~~~~~~~~~|~Socket Interface|~~~~~~~~~~~
|                 v

+-------------------------------+
|             MPTCP             |
| Subflow (TCP) | Subflow (TCP) |
+-------------------------------+
|       IP      |      IP       |
+-------------------------------+

MPTCP protocol stack

In general, MPTCP affects all interfaces that rely on the coupling of a TCP connection to a single IP address and TCP port pair, to one sockets endpoint, to one network interface, or to a given path through the network.

A design objective of MPTCP is that applications can continue to use the established sockets API without any changes. Still, some aspects have to be taken into account: In MPTCP, there is a one-to-many mapping between the socket endpoint and the subflows. As a consequence, the existing sockets interface functions cannot configure each subflow individually. In order to be backward compatible, existing APIs therefore should apply to all subflows within one connection, as far as possible.

4.2. Impact on the Use of Socket Options

The sockets API includes options that modify the behavior of sockets and their underlying communications protocols. Various socket
options exist on socket, TCP, and IP level. The value of an option can usually be set by the setsockopt() system function. The getsockopt() function gets information.

One commonly used TCP socket option (TCP_NODELAY) disables the Nagle algorithm as described in [2]. This option is also specified in the Posix standard [5]. Applications can use this option in combination with MPTCP exactly in the same way. It then disables the Nagle algorithm for the MPTCP connection, i.e., all subflows.

TODO: Setting this option could also trigger a different path scheduler algorithm - specifically, that which is designed for latency-sensitive traffic, as described in a later section.

Applications can also explicitly configure send and receive buffer sizes by the sockets API (SO_SNDBUF, SO_RCVBUF). These socket options can also be used in combination with MPTCP and then affect the buffer size of the MPTCP connection. However, when defining buffer sizes, application programmers should take into account that the transport over several subflows requires a certain amount of buffer for resequencing. Therefore, it does not make sense to use MPTCP in combination with very small receive buffers. Small send buffers may prevent MPTCP from efficiently scheduling data over different subflows.

It is assumed that any application that binds to INADDR_ANY does not care which addresses are in use locally, and so MPTCP can freely set up multiple subflows on such a connection. If an application uses a specific address, or sets the SO_BINDTODEVICE socket option to bind to a specific interface, then MPTCP MUST respect this and not interfere in the application’s choices. The extended sockets API will allow applications to express such preferences in an MPTCP-compatible way (e.g. bind to a subset of devices only).

Some network stacks also provide other implementation-specific socket options that affect TCP’s behavior. If a network stack supports MPTCP, it must be ensured that these options do not interfere.

4.3. Impact on Existing Other System-wide Settings

TODO: Socker buffer dimensioning: Requirement of larger resequencing buffer space

TODO: Could also affect interface configuration, information in local routing table, buffer management, etc.
4.4. Impact on Existing API Calls

There is an issue, to be resolved, regarding what data should be returned on a getpeername() or getsockname() request on the socket, i.e. to retrieve the IP address of the peer or of the local socket. Our initial thinking is that it should return the IP address pair that was first connected to, in all circumstances, even if that particular subflow is no longer in use. MPTCP-aware applications can use new API calls, documented later, in order to retrieve the full list of address pairs for the subflows in use.

4.5. Impact on Existing Sockets API Enhancements

The use of MPTCP can interact with various related sockets API extensions:

- SHIM API [7]: This API specifies sockets API extensions for the multihoming shim layer. TODO: Potential interactions will be addressed in a future revision of this memo.

- HIP API [8]: The Host Identity Protocol (HIP) also results in a new API. TODO: Potential interactions will be addressed in a future revision of this memo.

- IPv6 API [6]: The API for IPv6 leaves open the interaction with TCP.

5. Application Requirements

5.1. MPTCP Usage Scenarios

Applications that use TCP have different requirements on the transport layer. While developers have become used to the characteristics of regular TCP, new opportunities created by MPTCP could allow the service provided to be optimised further.

An application that wishes to transmit bulk data will want MPTCP to provide a high throughput service immediately, through creating and maximising utilisation of all available subflows. This is the default MPTCP use case.

But at the other extreme, there are applications that are highly interactive, but require only a small amount of throughput, and these are optimally served by low latency and jitter stability. In such a situation, it would be preferable for the traffic to use only the lowest latency subflow (assuming it has sufficient capacity), with one or two additional subflows for resilience and recovery purposes.
The choice between these two options affects the scheduler in terms of whether traffic should be, by default, sent on one subflow or across both. Even if the total bandwidth required is less than that available on an individual path, it is desirable to spread this load to reduce stress on potential bottlenecks, and this is why this method should be the default. It is recognised, however, that this may not benefit all applications that require latency/jitter stability, so the other (single path) option is provided.

In the case of the latter option, however, a further question arises: should additional subflows be used whenever the primary subflow is overloaded, or only when the primary path fails (hot-standby)? In other words, is latency stability or bandwidth more important to the application?

We therefore divide this option into two: Firstly, there is the single path which can overflow into an additional subflow; and secondly there is single-path with hot-standby, whereby an application may want an alternative backup subflow in order to improve resilience. In case that data delivery on the first subflow fails, the data transport could immediately be continued on the second subflow, which is idle otherwise.

In summary, there are three different "application profiles" concerning the use of MPTCP:

1. Bulk data transport
2. Latency-sensitive transport (with overflow)
3. Latency-sensitive transport (hot-standby)

These different application profiles affect both the management of subflows, i.e., the decisions when to set up additional subflows to which addresses as well as the assignment of data (including retransmissions) to the existing subflows. In both cases different policies can exist.

These profiles have been defined to cover the common application use cases. It is not possible to cover all application requirements, however, and as such applications should additionally have finer control over subflow and scheduling should they require. Requirements are TBD.

Although it is intended that such functionality will be achieved through new MPTCP-specific options, it may also be possible to infer some application preferences from existing socket options, such as TCP_NODELAY. Whether this would be reliable, and indeed appropriate,
5.2. Requirements on API Extensions

Because of the importance of the sockets interface there are several fundamental design objectives for the interface between MPTCP and applications:

- Consistency with existing sockets APIs must be maintained. In order to support the large base of applications using the original API, an application must be able to continue to use all standard socket interface functions when run on a system supporting MPTCP.

- Sockets API extensions must be minimized and independent of an implementation.

- The interface should both handle IPv4 and IPv6.

The following is a list of specific requirements from applications:

TODO: This list of requirements is preliminary and requires further discussion. Some requirements have to be removed.

REQ1: Turn on/off MPTCP: An application should be able to request to turn on or turn off the usage of MPTCP. This means that an application should be able to explicitly request the use of MPTCP if this is possible. Applications should also be able to request not to enable MPTCP and to use regular TCP transport instead. (This can be implicit in many cases, e.g., by the use of binding to a specific address versus all addresses).

REQ2: An application will want to be able to restrict MPTCP to binding to a given set of addresses or interfaces.

REQ3: An application should be able to know if multiple subflows are in use.

REQ4: An application should be able to extract a unique identifier for the connection (per endpoint), analogous to a port, i.e. it should be able to retrieve MPTCP’s connection identifier.

REQ5: An application should be able to enumerate all subflows in use, obtain information on the addresses used by a subflow, and obtain a subflow’s usage (e.g., ratio of traffic sent via this subflow).
REQ6: Set/get application profile, as discussed in the previous section.

REQ7: Constrain the maximum number of subflows to be used by an MPTCP connection. (Or just infer from application profile?)

REQ8: Request a change in scheduling between subflows? (i.e. a more granular version of application profile?)

REQ9: Request a change in the number of subflows in use, thus triggering removal or addition of subflows. (A finer control granularity would be: Request the establishment of a new subflow to a provided destination, and request the termination of a specified, existing subflow.)

REQ10: Control automatic establishment/termination of subflows? There could be different configurations of the path manager, e.g., ‘try ASAP’, ‘wait until there is a bunch of data, etc. (Tied to application profile?)

REQ11: Set/get preferred subflows or subflow usage policies? There could be different configurations of the multipath scheduler, e.g., ‘all-or-nothing’, ‘overflow’, etc. (Again, tied to application profile).

REQ12: Set/get sporadic sending of segments on unused paths ("keepalives").

REQ13: An application should be able to modify the MPTCP configuration while communication is ongoing, i.e., after establishment of the MPTCP connection.

6. Specification of API Extensions for MPTCP

6.1. Design Considerations

Multipath transport results in many degrees of freedom. MPTCP manages the data transport over different subflows automatically. By default, this is transparent to the application. But applications can use the sockets API extensions defined in this section to interface with the MPTCP layer and to control important aspects of the MPTCP implementation’s behaviour. The API uses non-mandatory socket options and is designed to be as light-weight as possible.

MPTCP mainly affects the sending of data. Therefore, most of the new socket options must be set in the sender side of a data transfer in order to take effect. TODO: Any control on the receiver side?
As this document specifies sockets API extensions, it is written so that the syntax and semantics are in line with the Posix standard \[5\] as much as possible.

6.2. Overview of Sockets Interface Extensions

The extended MPTCP API consist of several new socket options that are specific to MPTCP. All of these socket options are defined at TCP level (IPPROTO_TCP). These socket options can be used either by the getsockopt() or by the setsockopt() system call.

- TCP_MP_ENABLE: MPTCP enabled/disabled
- TCP_MP_MAXSUBFLOWS: Get/set maximum number of paths
- ...

TODO: Table of socket options

6.3. Detailed Description

6.3.1. TCP_MP_ENABLE

TODO: Description

6.3.2. TCP_MP_MAXSUBFLOW

TODO: Description

6.4. Usage examples

TODO: Example C code for one or more API functions

6.5. Discussion of Interactions

TODO: Some of the socket options defined in this document are overlapping with existing sockets API and care should be taken for the usage not to confuse with the overlapping features.

TODO: Interactions with system-wide settings?

6.6. Advice to Application Developers

TODO: E. g. use primary addresses and connection identifiers in a tuple instead of the traditional 5-tuple
7. Security Considerations

Will be added in a later version of this document.

8. IANA Considerations

No IANA considerations.

9. Conclusion

This document discusses MPTCP’s application implications and specifies an extended API. From an architectural point of view, MPTCP offers additional degrees of freedom concerning the transport of data. The extended sockets API allows applications to have additional control of some aspects of the MPTCP implementation’s behaviour and to obtain information about its usage. The new socket options for MPTCP can be used by getsockopt() and/or setsockopt() system calls. But it is also ensured that the existing sockets API continues to work.

10. Acknowledgments

Michael Scharf is supported by the German-Lab project (http://www.german-lab.de/) funded by the German Federal Ministry of Education and Research (BMBF). Alan Ford is supported by Trilogy (http://www.trilogy-project.org/), a research project (ICT-216372) partially funded by the European Community under its Seventh Framework Program. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

11. References

11.1. Normative References


11.2. Informative References


Authors’ Addresses

Michael Scharf
Alcatel-Lucent Bell Labs
Lorenzstrasse 10
70435 Stuttgart
Germany
EMail: michael.scharf@alcatel-lucent.com

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