Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: April 29, 2010

A. Ford
Roke Manor Research
C. Raiciu
M. Handley
University College London
October 26, 2009

TCP Extensions for Multipath Operation with Multiple Addresses
draft-ford-mptcp-multiaddressed-02

Status of this Memo

Copyright Notice

Abstract

TCP/IP communication is currently restricted to a single path per

connection, yet multiple paths often exist between peers. The simultaneous use of these multiple paths for a TCP/IP session would improve resource usage within the network, and thus improve user experience through higher throughput and improved resilience to network failure.

Multipath TCP provides the ability to simultaneously use multiple paths between peers. This document presents a set of extensions to traditional TCP to support multipath operation. The protocol offers the same type of service to applications as TCP - reliable bytestream - and provides the components necessary to establish and use multiple TCP flows across potentially disjoint paths.

Table of Contents

1. Introduction

Multipath TCP (henceforth referred to as MPTCP) is set of extensions to regular TCP [2] to allow a transport connection to operate across multiple paths simultaneously. This document presents the protocol changes required by Multipath TCP, specifically those for signalling and setting up multiple paths ("subflows"), managing these subflows, reassembly of data, and termination of sessions. This is not the only information required to create a Multipath TCP implementation, however. This document is complemented by several others:

- Architecture [3], which explains the motivations behind Multipath TCP and a functional separation through which an extensible MPTCP implementation can be developed.

- Congestion Control [4], presenting a safe congestion control algorithm for coupling the behaviour of the multiple paths in order to "do no harm" to other network users.

- Application Considerations [5], discussing what impact MPTCP will have on applications, what applications will want to do with MPTCP, and as a consequence of these factors, what API extensions an MPTCP implementation should present.

1.1. Design Assumptions

In order to limit the potentially huge design space, the authors imposed two key constraints on the multipath TCP design presented in this document:

- It must be backwards-compatible with current, regular TCP, to increase its chances of deployment

- It can be assumed that one or both endpoints are multihomed and multiaddressed

To simplify the design we assume that the presence of multiple addresses at an endpoint is sufficient to indicate the existence of multiple paths. These paths need not be entirely disjoint: they may share one or many routers between them. Even in such a situation making use of multiple paths is beneficial, improving resource utilisation and resilience to a subset of node failures. The congestion control algorithms as discussed in [4] ensure this does not act detrimentally.

There are three aspects to the backwards-compatibility listed above:

External Constraints: The protocol must function through the vast majority of existing middleboxes such as NATs, firewalls and proxies, and as such must resemble existing TCP as far as possible on the wire. Furthermore, the protocol must not assume the segments it sends on the wire arrive unmodified at the destination: they may be split or coalesced; options may be removed or duplicated.

Application Constraints: The protocol must be usable with no change to existing applications that use the standard TCP API (although it is reasonable that not all features would be available to such legacy applications).

Fall-back: The protocol should be able to fall back to standard TCP with no interference from the user, to be able to communicate with legacy hosts.

Areas for further study:

- In theory, since this is purely a TCP extension, it should be possible to use MPTCP with both IPv4 and IPv6 on dual-stack hosts, thus having the additional possible benefit of aiding transition.

- Some features of the design presented here could be extended to work with non-multi-addressed hosts by using other packet metadata (such as ports or flow label), packet marking, or partial (potenitally proxied) multipath.

1.2. Layered Representation

MPTCP operates at the transport layer, and its existence aims to be transparent to both higher and lower layers. It is a set of additional features on top of standard TCP, and as such MPTCP is designed to be usable by legacy applications with no changes. A possible implementation would be for such a feature to be a system-wide setting: "Use multipath TCP by default? Y/N". Multipath-aware applications would be able to use an extended sockets API to have further influence on the behaviour of MPTCP. Figure 1 illustrates this layering.

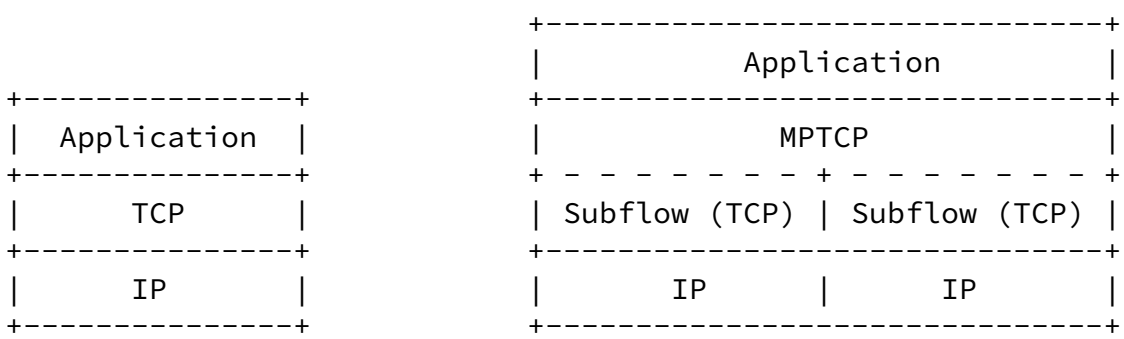


Figure 1: Comparison of Standard TCP and MPTCP Protocol Stacks

Detailed discussion of an architecture for developing a multipath TCP implementation, especially regarding the functional separation by which different components should be developed, is given in [3].

1.3. Operation Summary

This section provides a high-level summary of normal operation of MPTCP, and is illustrated by the scenario shown in Figure 2. A detailed description of operation is given in Section 4.

- To a non-MPTCP-aware application, MPTCP will be indistinguishable from normal TCP. All MPTCP operation is handled by the MPTCP implementation, although extended APIs could provide additional control and influence [5]. An application begins by opening a TCP socket in the normal way.

- An MPTCP connection begins as a single TCP session. This is illustrated in Figure 2 as being between Addresses A1 and B1 on Hosts A and B respectively.

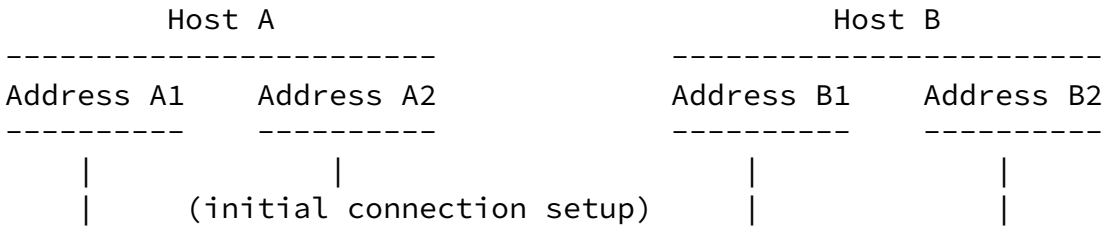
- If extra paths are available, additional TCP sessions are created on these paths, and are combined with the existing session, which continues to appear as a single connection to the applications at both ends. The creation of the additional TCP session is illustrated between Address A2 on Host A and Address B1 on Host B.

- MPTCP identifies multiple paths by the presence of multiple addresses at endpoints. Combinations of these multiple addresses equate to the additional paths. In the example, other potential paths that could be set up are A1<->B2 and A2<->B2. Although this additional session is shown as being initiated from A2, it could equally have been initiated from B1.

- The discovery and setup of additional TCP sessions (termed 'subflows') will be achieved through a path management method. This document describes a mechanism by which an endpoint can

initiate new subflows by using its additional addresses, or by signalling its available addresses to the other endpoint.

- MPTCP adds connection-level sequence numbers in order to reassemble the data stream in-order from multiple subflows. Connections are terminated by connection-level FIN packets as well as those relating to the individual subflows.



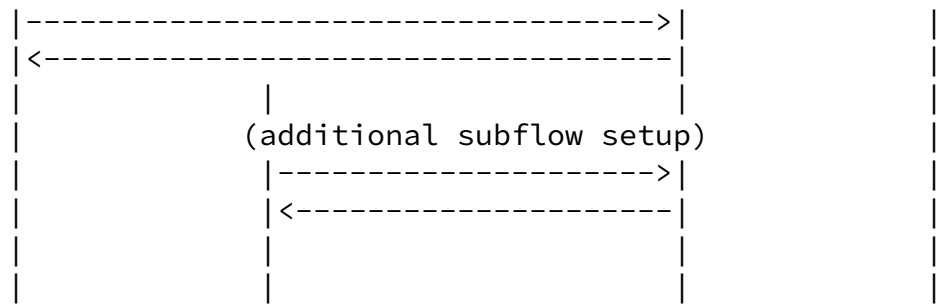


Figure 2: Example MPTCP Usage Scenario

1.4. Open Issues

This specification is a work-in-progress, and as such there are many issues that are still to be resolved. This section lists many of the key open issues within this specification; these are discussed in more detail in the appropriate sections throughout this document.

- Best handshake mechanisms (Section 4.1). This document contains a proposed scheme by which connections and subflows can be set up. It is felt that, although this is "no worse than regular TCP", there could be opportunities for significant improvements in security that could be included (potentially optionally) within this protocol.

- Issues around simulataneous opens, where both ends attempt to create a new subflow simultaneously, need to be investigated and behaviour specified.

- Appropriate mechanisms for controlling policy/priority of subflow usage (specifically regarding controlling incoming traffic, Section 4.4.3). The ECN signal is currently proposed but other alternatives, including per subflow receive windows or path

property options, could be employed instead.

- How much control do we want over subflows from other subflows (e.g. closing when interface has failed)? Do we want to differentiate between subflows and addresses (Section 4.2)?

- Do we want a connection identifier in every packet? E.g. would make implementation of IDS much easier?

- Best way of ensuring data/subflow sequence numbering mapping through middleboxes (Section 4.4)?

- Is there any benefit to a data-level acknowlegement?

1.5. 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 [1].

2. Terminology

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

Subflow: A stream of TCP packets sent over a path. A subflow is a component part of a connection between two endpoints.

Connection: A collection of one or more subflows, over which an application can communicate between two endpoints. There is a one-to-one mapping between a connection and a socket.

Token: A unique identifier given to a multipath connection by an endpoint. May also be referred to as a "Connection ID".

Endpoint: A host operating an MPTCP implementation, and either initiating or terminating a MPTCP connection.

3. Semantic Issues

In order to support multipath operation, the semantics of some TCP components have changed. To aid clarity, this section collects these semantic changes as a reference.

Sequence Number: The (in-header) TCP sequence number is subflow-

specific. To allow the receiver to reorder application data, an additional data-level sequence space is used. In this space, the initial SYN and the final DATA FIN occupy one octet. There is an explicit mapping of data sequence space to subflow sequence space, which is signalled through TCP options in data packets.

Receive Window: The receive window exists at the connection level, rather than at the subflow level, as it tries to regulate the sending rate of the sender to a slower receiver. With multipath TCP, each subflow MUST report the same global receive window, describing the per connection receive buffer.

FIN: The FIN only applies to a subflow, not to a connection. For a connection-level FIN, use the DATA FIN option.

ACK: The ACK acknowledges the subflow sequence number only, and the mapping to the data sequence number is handled out-of-band.

RST: The RST only applies to a subflow. There is no connection-level RST, since it would be impossible to distinguish the two, i.e. if there is no state about a subflow, the host cannot know to what connection the subflow is related. A connection is considered reset if every subflow sends a RST in response.

Address List: The address management is handled per-connection to permit the application of per-connection local policy.

5-tuple: The 5-tuple (protocol, local address, local port, remote address, remote port) presented to the application layer in a non-multipath-aware application is that of the first subflow, even if the subflow has since been closed and removed from the connection. These API issues are discussed in more detail in [5].

4. MPTCP Protocol

This section describes the operation of the MPTCP protocol, and is subdivided into sections for each key part of the protocol operation.

All MPTCP operations are signalled using optional TCP header fields. These TCP Options will have option numbers allocated by IANA, as listed in Section 9, and are defined throughout the following subsections.

4.1. Session Initiation

Session Initiation begins with a SYN, SYN/ACK exchange on a single path. Each of these packets will additionally feature the Multipath Capable TCP option (Figure 3), which declares the sender's locally unique 32-bit token for this connection, and contains a version field.

The "Multipath Capable" option declares an endpoint to be capable of operating Multipath TCP (or rather, more accurately, a desire to operate Multipath TCP on this particular connection). As well as this declaration, this field presents a token, which is used when adding additional subflows to this connection.

This token is generated by the sender and has local meaning only, hence it MUST be unique for the sender. The token MUST be difficult for an attacker to guess, and thus it is recommended it SHOULD be generated randomly. (However, see further discussions about security in Section 5, including the possibility of 64-bit tokens.)

This option is only present in packets with the SYN flag set. It is only used in the first TCP session of a connection, in order to identify the connection; all following connections will use path management options (see Section 4.2) to join the existing connection.

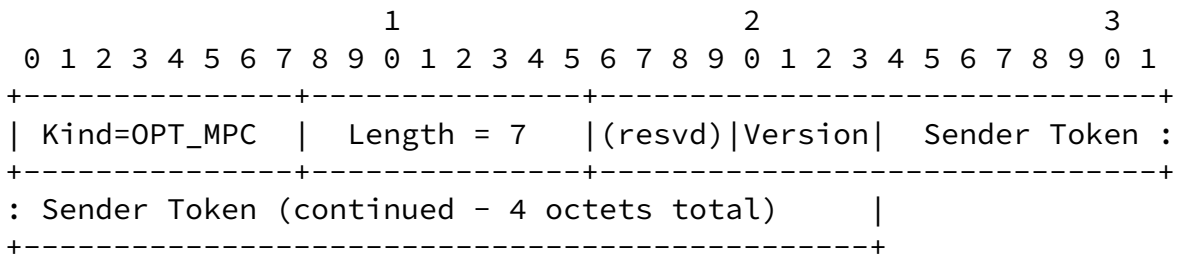


Figure 3: Multipath Capable option

The version field represents the version of MPTCP in use. The version provided in this specification is 0. The reserved bits may be used for connection-specific flags in later versions, or may be used to indicate an authentication method.

If a SYN contains a "multipath capable" option but the SYN/ACK does not, it is assumed that the recipient is not multipath capable and thus the MPTCP session will operate as regular, single-path TCP. If a SYN does not contain a "multipath capable" option, the SYN/ACK MUST NOT contain one in response.

If these packets are unacknowledged, it is up to local policy to decide how to respond. It is expected that a sender will eventually

fall back to single-path TCP (i.e. without the Multipath Capable Option), in order to work around middleboxes that may drop packets with unknown options; however, the number of multipath-capable attempts that are made first will be up to local policy. In the case of out-of-order packets, i.e. if a multipath-capable SYN/ACK is received in response to a multipath-capable SYN, after a standard SYN has been sent, then once again it is up to the initiator to choose how to behave. For example, it could respond to new connections using the previously declared token, or it could simply drop any new multipath options within the flow.

If an endpoint is known to be multiaddressed (e.g. through multiple addresses returned in a DNS lookup), alternative destination addresses SHOULD be tried first, before falling back to regular TCP.

In addition to this option, a Data Sequence Number option (discussed in Section 4.4) is included to provide an initial data-level sequence number (and this initial SYN counts as one octet in this space, as for a regular SYN in single-path TCP). This could also have some (minor) security benefits, discussed in Section 5.

4.2. Starting a New Subflow

Endpoints have knowledge of their own address(es), and can become aware of the other endpoint's addresses through signalling exchanges as described in Section 4.3. Using this knowledge, an endpoint will initiate a new subflow over a currently unused pair of addresses.

A new subflow is started as a normal TCP SYN/ACK exchange. The "Join" TCP option (Figure 4) is used to identify of which connection the new subflow should become a part. The token used is the locally unique token of the destination for the subflow, as defined by the Multipath Capable option received in the first SYN/ACK exchange.

It should be noted that, in theory, additional subflows can exist between any pair of ports; no explicit accept calls or bind calls are required to open additional subflows. To associate a new subflow to an existing connection, the token supplied in the subflow's SYN exchange is used for demultiplexing. This means that port numbers on

subflow SYN exchanges are not important, and any values can be used, as long as the 5-tuple is unique for each host. In practice, it is envisaged that most new subflows will connect to a port that is already in use as the source or destination port of an existing subflow, in order to have a greater chance of getting through firewalls and other middleboxes, and to support traffic engineering of the flows.

Deumultiplexing subflow SYNs MUST be done using the token; this is

unlike traditional TCP, where the destination port is used for demultiplexing SYN packets. Once a subflow is setup, demultiplexing packets is done using the five-tuple, as in traditional TCP.

The "Join" option includes an "Address ID". This is an identifier, locally unique to the sender of this option, and with only per-connection relevance, which identifies the source address of this packet. This serves two purposes. Firstly, if an address becomes unexpectedly unavailable on the sender, it can signal this to the receiver via a remove address option (Section 4.3.2) without needing to know what the source address actually is (thus allowing the use of NATs). Secondly, it allows correlation between new connection attempts and address signalling (Section 4.3.1), to prevent duplicate subflow initiation.

TBD: Instead of an Address ID, are there any cases where a Subflow ID (i.e. unique to the subflow) would be useful instead? For example, two addresses which become NATted to the same address?

This option can only be present when the SYN flag is set.

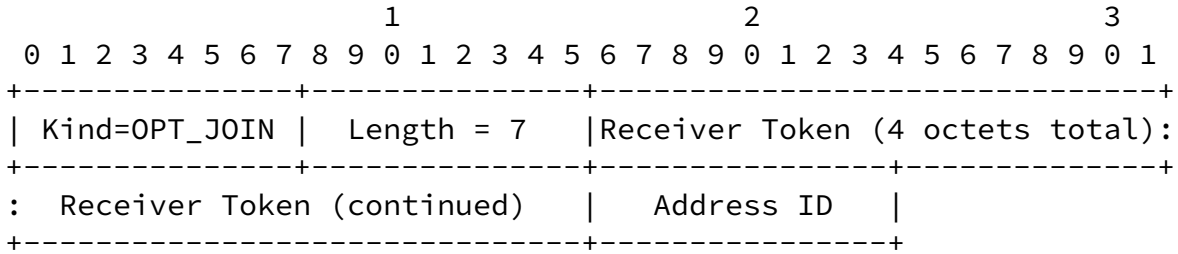


Figure 4: Join Connection option

4.3. Address Knowledge Exchange (Path Management)

We use the term "path management" to refer to the exchange of information about additional paths between endpoints, which in this design is managed by multiple addresses at endpoints. For more detail of the architectural thinking behind this design, see the separate document [3].

This design makes use of two methods of sharing such information, used simultaneously. The first is the direct setup of new subflows, already described in Section 4.2, where the initiator has an additional address. The second method is described in the following subsections, whereby addresses are signalled explicitly to the other endpoint, to allow it to initiate new connections. This approach, of two complementary mechanisms, has been chosen to allow addresses to change in flight, and thus support operation through NATs, whilst also allowing the signalling of previously unknown addresses, such as

those belonging to other address families (e.g. IPv4 and IPv6).

Here is an example of typical operation of the protocol:

- An endpoint that is multihomed starts an additional TCP session to an address/port pair that is already in use on the other endpoint, using a token to identify the flow (Section 4.2). (A multihomed destination may open a new subflow from its new address to an existing subflow's source address and port, or a multihomed source may open a new subflow from its new address to an existing subflow's destination and port).

- To expand upon this, say a connection is intiated from host "A" on (address, port) combination A1 to destination (address, port) B1 on host "B". If host A is multihomed, it starts an additional connection from new (address, port) A2 to B1, using B's previously declared token. Alternatively, if B is multhomed, it will try to set up a new TCP connection from B2 to A1, using A's previously declared token.

- Simultaneously (or after a timeout), an "Add Address" option (Section 4.3.1) is sent on an existing subflow, informing the receiver of the sender's alternative address(es). The recipient can use this information to open a new subflow to the sender's additional address. Using the previous notation, this would be an

Add Address packet sent from A1 to B1, informing B of address A2.

- The mix of using the SYN-based option and the Add Address option, including timeouts, is implementation-specific and can be tailored to agree with local policy.

- If host B successfully receives the first SYN, starting a new subflow, it can use the Address ID in the Join option to correlate this with the Add Address option that will also arrive on an existing subflow. Assuming the endpoint has already responded to the SYN with a SYN/ACK, it will know to ignore the Add Address option. Otherwise, if it has not received such a SYN, it will try to initiate a new subflow from one or more of its addresses to address A2 (triggered by the Add Address option). This is intended to permit new sessions to be opened if one endpoint is behind a NAT. A slight security improvement can be gained if a host ensures there is a correlated Add Address option before responding to the SYN.

Other scenarios are valid, however, such as those where entirely new addresses are signalled, e.g. to allow an IPv6 and an IPv4 path to be used simultaneously.

4.3.1. Adding Addresses

The Add Address TCP Option announces additional addresses on which an endpoint can be reached (Figure 5), which allows several (ID, address) pairs to be announced to the other endpoint. Multiple addresses can be added if there is sufficient TCP option space, otherwise multiple TCP messages containing this option will be sent. This option can be used at any time during a connection, depending on when the sender wishes to enable multiple paths and/or when paths become available.

Every address has an ID which can be used for address removal, and therefore endpoints must cache the mapping between ID and address. This is also used to identify Join Connection options (Section 4.2) relating to the same address, even when address translators are in use. The ID must be unique to the sender and connection, per address, but its mechanism for allocating such IDs is implementation-specific.

This option is shown for IPv4. For IPv6, the IPVer field will read 6, and the length of the address will be 16 octets not 4, and thus the length of the option will be 2 + (18 * number_of_entries). If there is sufficient TCP option space, multiple addresses can be included, with an ID following on immediately from the previous address, and their existance can be inferred through the option length and version fields.

NB: by having a IPVer field, we get four free reserved bits. These could be used in later versions of this protocol for expressing sender policy, e.g. one bit for "use now" or similar, to differentiate between subflows for backup purposes and those for throughput.

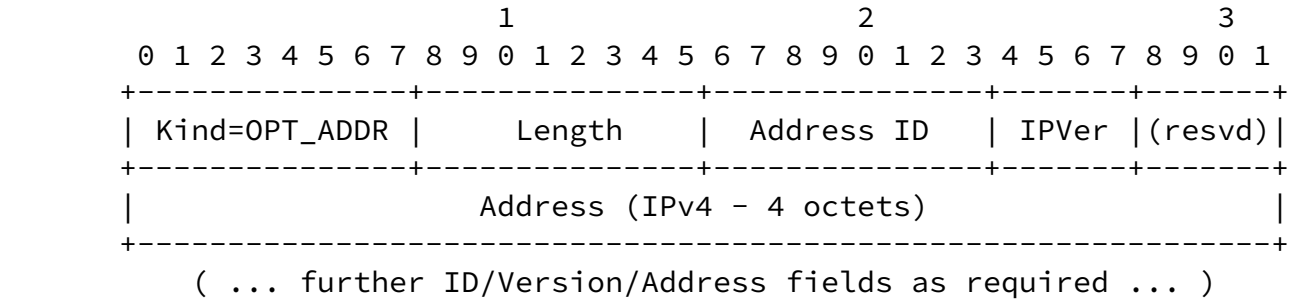


Figure 5: Add Address option (for IPv4)

4.3.2. Remove Address

If, during the lifetime of a MPTCP connection, a previously-announced address becomes invalid (e.g. if the interface disappears), the affected endpoint should announce this so that the other endpoint can remove subflows related to this address.

This is achieved through the Remove Address option (Figure 6), which will remove a previously-added address (or list of addresses) from a connection and terminate any subflows currently using that address.

The sending and receipt of this message should trigger the sending of FINs by both endpoints on the affected subflow(s) (if possible), as a courtesy to cleaning up middlebox state, but endpoints may clean up their internal state without a long timeout.

Address removal is undertaken by ID, so as to permit the use of NATs and other middleboxes. If there is no address at the requested ID, the receiver will silently ignore the request.

The standard way to close a subflow (so long as it is still functioning) is to use a FIN exchange as in regular TCP - for more information, see Section 4.5.

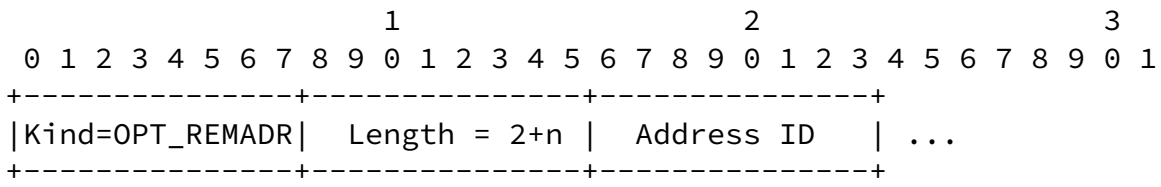


Figure 6: Remove Address option

4.4. General MPTCP Operation

This section discusses operation of MPTCP for data transfer. At a high level, an MPTCP implementation will take one input data stream from an application, and split it into one or more subflows. The data stream as a whole can be reassembled through the use of the Data Sequence Mapping (Figure 7) option, which defines the mapping from the data sequence number to the subflow sequence number. This is used by the receiver to ensure in-order delivery to the application layer. Meanwhile, the subflow-level sequence numbers (i.e. the regular sequence numbers in the TCP header) have subflow-only relevance.

The only acknowledgements are those at the subflow-level, so the sender must be able to map these acknowledgements to the data sequence numbers that were contained in the relevant packets. The

sender thus knows, if subflow data goes unackowledged, which part of the original data stream this equates to, and thus what data must be retransmitted. It is expected (but not mandated) that SACK [6] is used at the subflow level to improve efficiency.

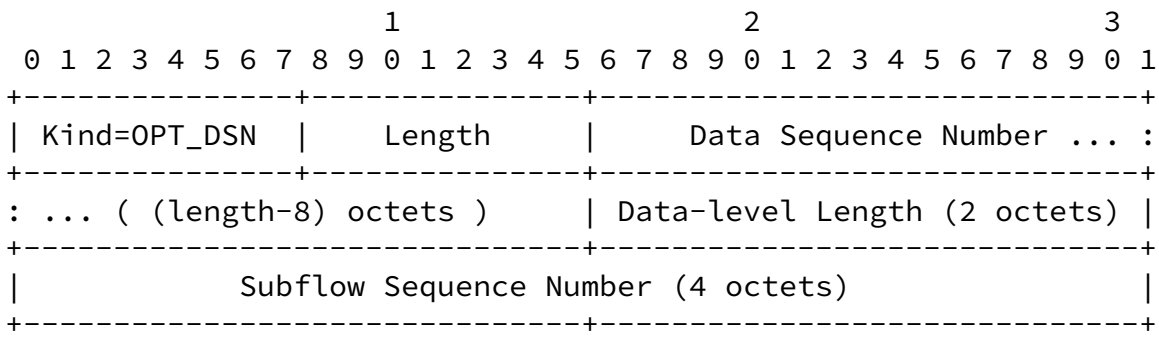


Figure 7: Data Sequence Mapping option

This option specifies a full mapping from data sequence number to subflow sequence number, informing the receiver that there is a one-to-one correspondence between the two sequence spaces for the specified length. The purpose of the explicit mapping is to assist with compatibility with situations where TCP/IP segmentation or coalescing is undertaken separately from the stack that is generating the data flow (e.g. through the use of TCP segmentation offloading on network interface cards, or by middleboxes such as performance enhancing proxies).

The data sequence number specified in this option is absolute, whereas the subflow sequence numbering is relative (the SYN at the start of the subflow has subflow sequence number 1).

A mapping is unique, in that the subflow sequence number is bound to the data sequence number after the mapping has been processed. It is not possible to change this mapping afterwards; however, the same data sequence number can be mapped on different subflows for retransmission purposes (see Section 4.4.4).

A receiver MUST NOT accept data for which it does not have a mapping to the data sequence space. To do this, the receiver will not acknowledge the unmapped data at subflow level. It is better to have a subflow fail than to accept data in the wrong order. However, if there was a lost packet in the subflow, the receiver SHOULD wait for this to be retransmitted before closing the subflow, since the lost packet may contain the necessary mapping information.

Although it is expected that initial implementations will use 32-bit data sequence numbers (i.e. 4 octets, so a length field of 12),

setting the length field to 16 and including a 64-bit sequence number (eight octets) MUST be considered valid and processed appropriately. This may have also have useful security implications, discussed in Section 5.

As with the standard TCP sequence number, the data sequence number should not start at zero, but at a random value to make session hijacking harder. This is done by including a Data Sequence Number option along with the Multipath Capable option in the initial SYN (which occupies one octet of data sequence space; see Section 4.1). In this case, to save option space, neither the data-level length nor the subflow sequence number fields are present in this option, so the Length field will be the length of the Data Sequence Number, plus two octets.

The Data Sequence Mapping does not need to be included in every MPTCP packet, as long as the subflow sequence space in that packet is covered by a mapping known at a receiver. This can be used to reduce overhead in cases where the mapping is known in advance; one such case is when there is a single subflow between the endpoints, another is when segments of data are scheduled in larger than packet-sized chunks.

The MPTCP data and subflow level sequence numbering could be seen to be analogous to that used in SACK, however there are subtle differences. The key similarity is that it is possible to have temporary "holes" in the received data sequence space - later data may have arrived earlier (most likely on a different subflow), but does not need to be retransmitted. The "holes" are later filled in. The key difference, however, is that while SACK can rely on the regular TCP cumulative acknowledgements to indicate how much data has been successfully received (with no holes), there is no similar method in MPTCP. Instead, the sender must keep track of the acknowledgements to derive what data has been successfully received. This leads to some oddities especially with session termination (see Section 4.5).

4.4.1. Receive Window Considerations

Regular TCP advertises a receive window in each packet, telling the sender how much data the receiver is willing to accept past the cumulative ack. The receive window is used to implement flow control, throttling down fast senders when receivers cannot keep up.

MPTCP also uses a unique receive window, shared between the subflows. The idea is to allow any subflow to send data as long as the receiver is willing to accept it; the alternative, maintaining per subflow

receive windows, could end-up stalling some subflows while others

would not use up their window.

An issue will arise regarding how large a receive buffer to implement. The lower bound would be the maximum bandwidth/delay product of all paths, however this could easily fill when a packet is lost on a slower subflow and needs to be retransmitted (see Section 4.4.4). The upper bound would be the maximum RTT multiplied by the maximum total bandwidth available. This will cover most eventualities, but could easily become very large. It is FFS what the best approach is.

4.4.2. Congestion Control Considerations

Different subflows in an MPTCP connection have different congestion windows. To achieve resource pooling, it is necessary to couple the congestion windows in use on each subflow, in order to push most traffic to uncongested links. One algorithm for achieving this is presented in [4]; the algorithm does not achieve perfect resource pooling but is "safe" in that it is readily deployable in the current Internet.

It is foreseeable that different congestion controllers will be implemented for MPTCP, each aiming to achieve different properties in the resource pooling/fairness/stability design space. Much research is expected in this area in the near future.

Regardless of the algorithm used, the design of the MPTCP protocol aims to provide the congestion control implementations sufficient information to take the right decisions; this information includes, for each subflow, which packets where lost and when.

4.4.3. Subflow Policy

Within a local MPTCP implementation, a host may use any local policy it wishes to decide how to share the traffic to be sent over the available paths.

In the typical use case, where the goal is to maximise throughput, all available paths will be used simultaneously for data transfer, using coupled congestion control as described in [4]. It is

expected, however, that other use cases will appear.

For instance, a possibility is an 'all-or-nothing' approach, i.e. have a second path ready for use in the event of failure of the first path, but alternatives could include entirely saturating one path before using an additional path (the 'overflow' case). Such choices would be most likely based on the monetary cost of links, but may also be based on properties such as the delay or jitter of links,

where stability is more important than throughput. Application requirements such as this are discussed in detail in [5].

The ability to make effective choices at the sender requires full knowledge of the path "cost", which is unlikely to be the case. There is no mechanism in MPTCP for a receiver to signal their own particular preferences for paths, but this is a necessary feature since receivers will often be the multihomed party, and may have to pay for metered incoming bandwidth. Instead of incorporating complex signalling, it is proposed to use existing TCP features to signal priority implicitly. If a receiver wishes to keep a path active as a backup but wishes to prevent data being sent on that path, it could stop sending ACKs for any data it receives on that path. The sender would interpret this as severe congestion or a broken path and stop using it. We do not advocate this method, however, since this is brutal, naive, and will result in unnecessary retransmissions.

Therefore, a proposal is to use ECN [7] to to provide fake congestion signals on paths that a receiver wishes to stop being used for data. This has the benefit of causing the sender to back off without the need to retransmit data unnecessarily, as in the case of a lost ACK. This should be sufficient to allow a receiver to express their policy, although does not permit a rapid increase in throughput when switching to such a path.

TBD: This is clearly an overload of the ECN signal, and as such other solutions, such as explicitly signalling path operation preferences (such as in the reserved bits of certain TCP options, or through entirely new options) may be a preferred solution.

4.4.4. Retransmissions

This protocol specification does not mandate any mechanisms for

handling retransmissions in the event of path failures, and much will be dependent upon local policy (as discussed in Section 4.4.3). The data sequence number, as given in a TCP option, is used to reassemble the incoming streams before presentation to the application layers, so a sender is free to re-send data with the same data sequence number on a different subflow. When doing this, an endpoint must still retransmit the original data on the original subflow, in order to preserve the subflow integrity (middleboxes could replay old data, and/or could reject holes in subflows), and a receiver will ignore these retransmissions. While this is clearly suboptimal, for compatibility reasons we feel this is the best behaviour. Optimisations could be negotiated in future versions of this protocol.

Of course, retransmissions on alternative subflows will only occur if

this is what local policy suggests. Indeed, it may be equally valid to retransmit on the same subflow if alternative paths have considerably worse quality of service, or are only kept for backup purposes. Additionally, it may be possible for some implementations to signal from lower layers if there are problems with the paths, and so more appropriate responses could occur.

4.5. Closing a Connection

Under single path TCP, a FIN signifies that the sender has no more data to send. In order to allow subflows to operate independently, however, and with as little change from regular TCP as possible, a FIN in MPTCP will only affect the subflow on which it is sent. This allows nodes to exercise considerable freedom over which paths are in use at any one time. The semantics of a FIN remain as for regular TCP, i.e. it is not until both sides have ACKed each other's FINs that the subflow is fully closed.

When an application calls close() on a socket, this indicates that it has no more data to send, and for regular TCP this would result in a FIN on the connection. For MPTCP, an equivalent mechanism is needed, and this is the DATA FIN. This option, shown in Figure 8, is attached to a regular FIN option on a subflow.

A DATA FIN is an indication that the sender has no more data to send, and as such can be used as a rapid indication of the end of data from

a sender. A DATA FIN, as with the FIN on a regular TCP connection, is a unidirectional signal.

The DATA FIN is an optimisation to rapidly indicate the end of a data stream and clean up state associated with a MPTCP connection, especially when some subflows may have failed. Specifically, when a DATA FIN has been received, IF all data has been successfully received, timeouts on all subflows MAY be reduced. Similarly, when sending a DATA FIN, once all data (including the DATA FIN, since it occupies one octet of data sequence space) has been acknowledged, FINs must be sent on every subflow. This applies to both endpoints, and is required in order to clean up state in middleboxes.

There are complex interactions, however, between a DATA FIN and subflow properties:

- A DATA FIN MUST only be sent on a packet which also has the FIN flag set.

- A DATA FIN occupies one octet (the final octet) of Data Sequence Number space. Therefore, even if there is no user data, a Data Sequence Number option MUST be added to a packet containing the

  DATA FIN option. This allows the receiver to easily determine the last data sequence number that should have been received.

- There is a one-to-one mapping between the DATA FIN and the subflow's FIN flag (and its associated sequence space and thus its acknowlegement). In other words, when a subflow's FIN flag has been acknowledged, the associated DATA FIN is also acknowledged.

- As such, the acknowledgement of a FIN and DATA FIN DOES NOT indicate that all data has been successfully received. Because the data level ack is inferred from subflow acks, an endpoint must use subflow acks to discover when all data up to and including the DATA FIN has been received.

It should be noted that an endpoint may also send a FIN on an individual subflow to shut it down, but this impact is limited to the subflow in question. If all subflows have been closed with a FIN, that is equivalent to having closed the connection with a DATA FIN.

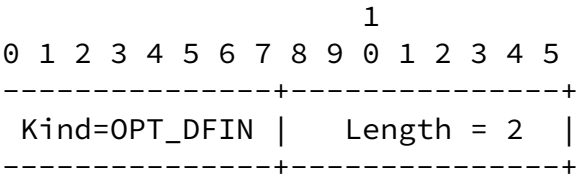


Figure 8: DATA FIN option

4.6. Error Handling

TBD

Unknown token in MPTCP SYN should equate to an unknown port, e.g. a TCP reset? We should make this as silent and tolerant as possible. Where possible, we should keep this close to the semantics of TCP. However, some MPTCP-specific issues such as where a data sequence number is missing from a subflow, will definitely need MPTCP-specific errors handling in those cases.

5. Security Considerations

TBD

(Token generation, handshake mechanisms, new subflow authentication, etc...)

A generic threat analysis for the addition of multipath capabilities to TCP is presented in [8]. The protocol presented here has been

designed to minimise or eliminate these identified threats. (A future version of this document will explicitly address the presented threats).

The development of a TCP extension such as this will bring with it many additional security concerns. We have set out here to produce a solution that is "no worse" than current TCP, with the possibility that more secure extensions could be proposed later.

The primary area of concern will be around the handshake to start new subflows which join existing connections. The proposal set out in Section 4.1 and Section 4.2 is for the initiator of the new subflow

to include the token of the other endpoint in the handshake. The purpose of this is to indicate that the sender of this token was the same entity that received this token at the initial handshake.

One area of concern is that the token could be simply brute-forced. The token must behard to guess, and as such could be randomly generated. This may still not be strong enough, however, and so the use of 64 bits for the token would alleviate this somewhat.

Use of these tokens only provide an indication that the token is the same as at the initial handshake, and does not say anything about the current sender of the token. Therefore, another approach would be to bring a new measure of freshness in to the handshake, so instead of using the initial token a sender could request a new token from the receiver to use in the next handshake. Hash chains could also be used for this purpose.

Yet another alternative would be for all SYN packets to include a data sequence number. This could either be used as a passive identifier to indicate an awareness of the current data sequence number (although a reasonable window would have to be allowed for delays). Or, the SYN could form part of the data sequence space - but this would cause issues in the event of lost SYNs (if a new subflow is never established), thus causing unnecessary delays for retransmissions.

6. Interactions with Middleboxes

TBD

How we get around NATs, firewalls. Problems with TCP proxies. How to make an MPTCP-aware middlebox, ...

7. Interfaces

TBD

Interface with applications, interface with TCP, interface with lower

layers...

Discussion of interaction with applications (both in terms of how MPTCP will affect an application's assumptions of the transport layer, and what API extensions an application may wish to use with MPTCP) are discussed in [5].

8. Acknowledgements

The authors are 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.

The authors gratefully acknowledge significant input into this document from many members of the Trilogy project, notably Iljitsch van Beijnum, Lars Eggert, Marcelo Bagnulo Braun, Robert Hancock, Pasi Sarolahti, Olivier Bonaventure, Toby Moncaster, Philip Eardley and Andrew McDonald.

9. IANA Considerations

This document will make a request to IANA to allocate new values for TCP Option identifiers, as follows:

Symbol	Name	Ref	Value
OPT_MPC	Multipath Capable	Section 4.1	(tbc)
OPT_ADDR	Add Address	Section 4.3.1	(tbc)
OPT_REMADR	Remove Address	Section 4.3.2	(tbc)
OPT_JOIN	Join Connection	Section 4.2	(tbc)
OPT_DSN	Data Sequence Number	Section 4.4	(tbc)
OPT_DFIN	DATA FIN	Section 4.5	(tbc)

Table 1: TCP Options for MPTCP

10. References

10.1. Normative References

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

10.2. Informative References

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

[3] Ford, A., Raiciu, C., Barre, S., Iyengar, J., and B. Ford, "Architectural Guidelines for Multipath TCP Development", draft-ford-mptcp-architecture-00 (work in progress), October 2009.

[4] Raiciu, C., Handley, M., and D. Wischik, "Coupled Multipath-Aware Congestion Control", draft-raiciu-mptcp-congestion-00 (work in progress), October 2009.

[5] Scharf, M. and A. Ford, "MPTCP Application Interface Considerations", draft-scharf-mptcp-api-00 (work in progress), October 2009.

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

[7] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[8] Bagnulo, M., "Threat Analysis for Multi-addressed/Multi-path TCP", draft-bagnulo-mptcp-threat-00 (work in progress), October 2009.

[9] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

Appendix A. Notes on use of TCP Options

The TCP option space is limited due to the length of the Data Offset field in the TCP header (4 bits), which defines the TCP header length in 32-bit words. With the standard TCP header being 20 bytes, this leaves a maximum of 40 bytes for options, and many of these may already be used by options such as timestamp and SACK.

As such, when doing address list manipulation, not all data may fit. This can be mitigated in one of two ways:

- Using an option to extend the option space, such as that proposed in [9], which proposes an option providing a 16-bit header length field. Such an option could only be used between nodes that support it, however, and so long options could not be used until a handshake is complete.

- Alternatively, since at least one IP address option field should be able to fit per packet, address list manipulation can be undertaken with one address per packet. One method could be to wait for data to send, and then append one new address per packet. This would seem reasonable if the TCP session begins rapidly, but if it is required that the multipath session is ready before the first data is to be sent, address list manipulation would be required on empty data (signalling only) packets. Issues may arise regarding acknowledged delivery of signalling versus data - this is discussed in Section 3 below.

Appendix B. Resync Packet

In earlier versions of this draft, we proposed the use of a "re-sync" option that would be used in certain circumstances when a sender needs to instruct the receiver to skip over certain subflow sequence numbers (i.e. to treat the specified sequence space as having been received and acknowledged).

The typical use of this option will be when packets are retransmitted on different subflows, after failing to be acknowledged on the original subflow. In such a case, it becomes necessary to move forward the original subflow's sequence numbering so as not to later transmit different data with a previously used sequence number (i.e. when more data comes to be transmitted on the original subflow, it would be different data, and so must not be sent with previously-used (but unacknowledged) sequence numbering).

The rationale for needing to do this is two-fold: firstly, when ACKs are received they are for the subflow only, and the sender infers from this the data that was sent - if the same sequence space could be occupied by different data, the sender won't know whether the

intended data was received. Secondly, certain classes of middleboxes may cache data and not send the new data on a previously-seen sequence number.

This option was dropped, however, since some middleboxes may get confused when they meet a hole in the sequence space, and do not

understand the resync option. It is therefore felt that the same data must continue to be retransmitted on a subflow even if it is already received after being retransmitted on another. There should not be a significant performance hit from this since the amount of data involved and needing to be retransmitted multiple times will be relatively small.

Therefore, it is necessary to 're-sync' the expected sequence numbering at the receiving end of a subflow, using the following TCP option. This packet declares a sequence number space (inclusive) which the receiving node should skip over, i.e. if the receiver's next expected sequence number was previously within the range start_seq_num to end_seq_num, move it forward to end_seq_num + 1.

This option will be used on the first new packet on the subflow that needs its sequence numbering re-synchronised. It will be continue to be included on every packet sent on this subflow until a packet containing this option has been acknowledged (i.e. if subflow acknowledgements exist for packets beyond the end sequence number). If the end sequence number is earlier than the current expected sequence number (i.e. if a resync packet has already been received), this option should be ignored.

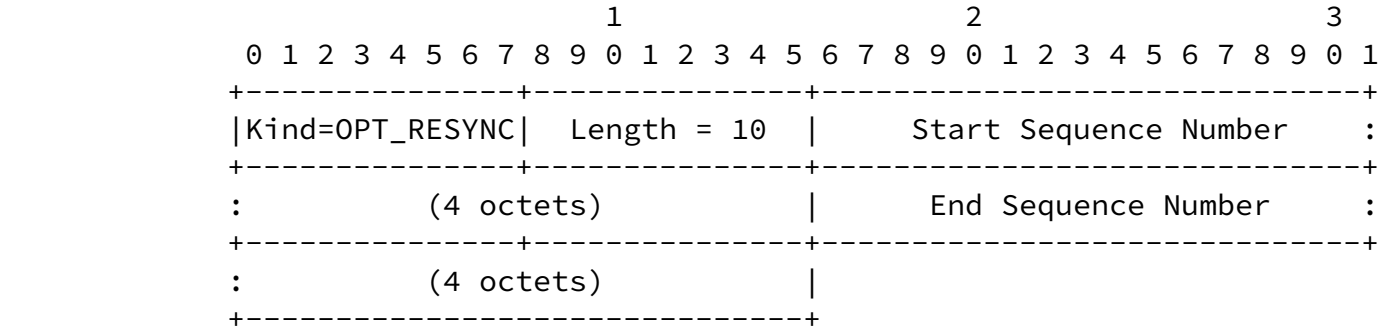


Figure 9: Resync option

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