# Multipath TCP Signalling

Costin Raiciu

February 9, 2010

## 1 Introduction

One design issue for multipath TCP is whether to use TCP options or the payload to signal multipath control information. This document analyzes the protocol design implications of these two choices, highlighting plusses and minuses for both approaches. We find that there are no show stoppers either way, and that both solutions seem "good enough" for deployment.

This document is structured as follows. We begin with a description of the multipath control information outlining its characteristics (e.g. size) and requirements (e.g. reliable, in order delivery). Then, we discuss design issues for both option-based and payload-based signalling. Finally, we provide a succint summary comparison between the two approaches.

## 2 Types of Control Information used by Multipath TCP

Multipath TCP connections need to signal control data to function properly. Control data includes mapping of data sequence numbers to subflow sequence numbers, receive window, policy information, address information, security information, and so forth.

**Data Sequence Mapping.** Multipath TCP relies on TCP as a subflow protocol: data from the sender is split across the existing subflows, where it is packed into segments and sent to the other end. At the receiver, the subflow part checks sequence numbers, etc., and passes the data up to the multipath layer. The multipath reorders the data segments and sends them to the application. To be able to do so, the multipath layer needs to know which data sequence numbers are associated with each packet.

Data sequence mappings include a subflow sequence number (4B), data sequence number (4-8B) and length (2B), yielding a total length of 10-14B.

Although the mapping could be sent on any subflow and at any time, it seems natural to "tie" the mapping information to the segment carying the data. The mapping is needed when and if the segment arrives.

The data sequence mapping will be sent to the other end at the same time segments are sent. The transmission must be reliable in that, if the segment arrives at the destination the data mapping must arrive at the destination too, or otherwise the destination does not know what to do with the packet.

Note that delivery of the mapping data must not be reliable in the general sense: every mapping ever sent does not need to eventually reach the other end. Say subflow 1 sends connection data 1-10 as segment 101-110, and the path fails before the segment arrives. The data 1-10 will be resent on subflow 2, as segment 201-210. The original mapping (101-110 $->$ 1-10) could be also resent on subflow 2, but its pointless to do so: the receiver only needs this information if it ever receives the 101-110 segment from subflow 1.

**Add/Remove Address.** In an MPTCP connection one or both subflows have multiple addresses. After initial subflow establishment, they can use their local addresses to setup additional subflows to the remote end. This may not work in all cases: for instance, if a server has an additional address it will not be able to connect to a NATed client.

In such cases, the server's MPTCP stack uses the "add address" control message to signal the other end it has an address available. The client can use this address to open a new subflow. The "add address" has mostly an informational status, and thus it appears to not require reliable delivery. However, there are cases when failing to signal an additional address may break the entire connection; one such case is when all the other subflows fail and the additional address could help the connection survive.

The reverse is to announce to the other end an IP address is no longer available, for instance because the corresponding interface went down. The other end will cleanup the state for subflows terminating on that interface.

Ordering matters, especially between add and remove messages. Say the server is mobile, gets a new interface, announces it, and then its old interface dies, and this is announced too. If the two options are processed in the right order, the connection survives; if they're processed the other way around the connection dies.

"Add address messages" contain a list of (id, protocol, address) structures. The id and protocol fields take 2B in total (12bits id, 4bit protocol), The addresses have 4B for IPv4 and 16B for IPv6, "Remove address" messages are small, as they only include the address ID.

**Data ACK.** At first glance, MPTCP can use subflow ACKs and locally saved mapping information to detect which data has arrived, without needing data ACKs.

However, a flurry of network-based performance enhancing proxies exist, and they actively intervene in an existing connection. It is possible that such a PEP acks a segment, but the path to the receiver fails without delivering the segment. The PEP will try to retransmit, but to no avail if the path is still failed. In the meantime, the sending MPTCP now thinks the associated data was received, yet the receiver will "close" the receive window waiting for the missing packet. MPTCP is stuck in this case, and cannot make progress despite having working paths.

To avoid such situations MPTCP includes a data ACK, which is a cumulative ack of data at the connection layer. As similar TCP ACKs, data ACKs do not need reliable delivery, and it is pointless to resend lost ACKs.

The data ACK has implications on the send buffer management. If it is not used, subflow acks will be used to free data from the sender's buffer. When the data ACK is used, the sender will use it to free packets from its send buffer. If it used the subflow ACK instead, it could release the segment before the destination actually received it (see example above).

**Data FIN.** The data FIN announces to the other end that the multipath connection has ended: no data will be flowing in that direction after this message. As such, the data fin must receive a data sequence number, and it must be mapped on a subflow. To make this easier, the data FIN must be sent on a FIN segment (that occupies sequence space already) so no other data.

**Security Information.** Multipath TCP may need to transfer security information to protect the connection from passive or active attackers, This needs reliable signalling, and, in contrast with the previous data, may be much larger. For instance, public keys typically have 1024-2048 bits (128-256B).

**Receive Window.** As standard TCP, multipath TCP must provide flow control to pace a sender that is sending faster than the receiver can consume. The design question is whether to use a connection receive window only, or a connection level receive window and per subflow receive windows. This choice affects the way the receive window is signalled. If there is a single connection receive window, it can just be signalled in each TCP segment on each subflow. If there are also subflow receive windows, the connection level window must be signalled otherwise.

Multiple receive windows seem to give little benefit (especially since policy information is transmitted using other options), yet they open the possibility of deadlocks if are not properly coordinated. That is why it seems better to advertise a single, connection-level receive window.

**Subflow Policy/Other stuff.** Multipath TCP needs to be able to name flows (combination of address IDs? what about network multipath?) and provide a preference level for each. Data gets sent only on the subflows with the highest preference.

# 3 Using TCP Options for Signalling

TCP options have been the traditional way to evolve TCP. They allow the use of 40B in each TCP header to signal arbitrary data to the remote end. Some of this space is already used by existing options, most notably by timestamp option (10B) and SACK (variable length, can use the whole available space).

**Option Space Limits.** One obvious issue is that the max available space is not big enough for the security extensions. We will show it is enough for most of the other options.

The data sequence number is necesary on data packets, not on ACKs. As flows are overwhelmingly asymetric, this means the forward path will carry little SACK information, and MPTCP effectively has 30B available for use. This accomodates all the options by themselves.

For security information, and for other types of information, it must be possible to add data in the TCP payload. One way to do this is to add a simple "control information" option stating the subflow sequence number and length of the control data, and embed the data in the payload. Note that this forces the data to be delivered reliably and in order.

**Ensuring Reliability.** The other problem is the lack of reliability for options: they can be stripped, modified or deleted by middleboxes. This is an issue for "Add/Remove address" messages and policy messages, and needs to be fixed. The obvious way is to add sequence numbers to control messages information, and to ack this information[1]. As a small number of these options may be in flight at any time, it seems 1B is sufficient as a sequence number. Also, because these messages are comparatively rare, the extra overhead is small.

Data sequence mapping messages do not need extra attention. If the segments also acked at connection level (i.e. there is a data ACK), we can extract from this an indication that both the segment and the mapping info was delivered.

If, however, the segment is not acked at connection level but is acked at subflow level, two things can happen: either the segment got lost after a middlebox crashed, or the segment was delivered but the mapping information wasn't (e.g. it was stripped from the headers). In both cases, MPTCP will retransmit both the segment and the data on another subflow. If a subflow consistently strips options, MPTCP will end up not using it. In contrast, if a middlebox strips options only on data packets, MPTCP with payload encoding will be able to use the path.

The data ACK, as regular ACKs, does not need reliable delivery. Losing a few of these packets is not normally an issue, as the cumulative ACK nicely summarizes the state of the connection.

Security negotiation typically happens at the beginning of the connection, and we can use payload reliability to ensure reliable, in-order data transmission.

# 4 Using TCP Payload for Signalling

With this approach, all the data is embedded in the payload of the subflows. Control message size is no longer a problem - arbitrary size messages can be embedded.

There are two basic ways to alternate both control and data messages in the payload. One is encoding using type-length-value, and the other is to encode type-value and use escape sequences to mark the end of control message. This second approach requires a linear scan of the segment at the receiver to find boundaries, requires escaping the escape sequences in the data, so it will be slower. We will use the first approach.

Subflow level reliable delivery is provided by default, for all control messages: every control message will get a sequence number and will be acked by the subflow ACK. The subtlety here is that we have to retransmit the same control data when it gets lost; we cannot update it, if we want to transmit more urgent data. This is because traffic normalization mandates that the same payload should be delivered for the same sequence number; hence, traffic normalizers cache the payload and will retransmit the cached data when they receive segments with cached sequence numbers. For the sender this means that if it replaces the payload on retransmitted packets and gets an ack for the data, it cannot be sure if the old or the new payload was received.

There is one particular control message that prefers timely delivery to reliability: data ACK. TCP ACKs are sent as side-effects of segments arriving at the receiver, an provide an ack-clock that allows the sender to safely send new segments in the network. The ACK stream is unreliable, and it is not congestion controlled. When ACK packets are lost, TCP recovers by the cumulative nature of ACKs; later ACKs will include strictly more information than the ones that were lost, so there is no need to retransmit.

With payload encoding, however, data ACKs are sent as regular payload: reliably, in order and are congestion controlled. If they are lost, they will be unnecesarily retransmitted despite the fact that newer ACKs might supersede them. Optimizations are possible here to override the ACK with the most recent

[1] Another way is to just echo the option once received. This works alright as long as there is a single option in flight.

one. In the worst case, this can result in timeouts at the sender, or unnecesary fast retransmits. It is unclear how bad this effect is.

If the reverse path has high loss rates the data ACKs will be throttled by the congestion window. This raises an interesting question: when do we send data ACKs? One way would be to piggyback on regular ACKs, but this would mean we would end up throttling the regular ACK stream. This, in effect, may make multipath subflows lose to NewReno subflows when the reverse path is lossy.

Thus, it seems data ACKs must be sent independently of subflow ACKs, or using a subset of subflow ACKs. The first option increases overall overhead, so the second option seems more appropriate.

**Future Middleboxes.** Payload encoding is likely to go through most middleboxes because it perfectly "hides" the multipath connection inside what looks like a regular TCP flow. This is good for getting through current middleboxes, but what does this mean for the future generation of middleboxes?

Multipath aware middleboxes will likely soon appear, and will try to optimize different aspects of multipath transmission. Because most of the control data is now in the payload, these middleboxes will have to parse the payload to find the control information. To cope with segmentation, they might hold state across TCP segments just to know where to start parsing (e.g. know where the next control information starts). Finally, if a middlebox sees packets of an ongoing subflow without seeing the SYN exchange (maybe because it rebooted, or the subflow was rerouted, or the middlebox does not maintain per connection information) it has no way of finding out those packets belongs to a multipath connection.

**Ordering.** Global ordering is not provided by default; for control messages that need global ordering, sequence numbers will need to be added inside the control message. Following the same reasoning as with data ACKs, it may be necesary to provide a connection level ACK for these control messages: otherwise, the ACK at subflow level can be misleading in showing that the message was received. Note that this is not a problem for data sequence mapping messages.

The data sequence mapping can be made 4B shorter if we use payload encoding, as the subflow sequence number does not have to be stated explicitly.

**Implementation.** Payload encoding seems to raise issues for the stack implementers. Typically, options were processed first to provide control data, then the payload was placed in the reorder/receive buffer. Now, the processing involves parsing the payload itself in two different ways, depending on which type of data arrived.

We observe, though, that this new type of processing needs to take place at connection level only, in the multipath subflow; at subflow level, processing remains unchanged. At the least, implementing multipath means implementing the connection level part of it, and reusing the existing TCP code for subflows.

As a matter of fact, it is simpler to "signal" the control options from the subflow to the connection layer if the payload is used to encode control messages, otherwise the stack will have to parse MPTCP-related options at subflow level and somehow transmit them to connection level. The same applies for transmitting control messages: the connection level must somehow instruct the subflow level to encode these control messages as options.

# 5 Comparison

Here we list all known differences between the two approaches, to facilitate a comparison.

| **Topic** | **Options** | **Payload** |
|---|---|---|
| MPTCP Capable | must use options | must use options |
| Token | must use options | must use options |
| Data Sequence Mapping<br>Size<br>Reliability | <br><br>10-14B<br>Uses Data ACK to infer reception of mapping and of segment. Assumes paths that strip options on data segments are failed, and will stop using them. | <br><br>6-8B<br>Still needs to use data ACK to guard against failure of end-to-end subflow ACKs. |

| | | |
|---|---|---|
| Data ACK | send as and shares fate of regular ACKs | data ACKs get resent until they are received. It will be space inneficient, especially if there is a high loss rate on the return path. Because of delayed acks, there may be quite a few RTOs on the return path. Optimizations are needed here to reduce the amount of useless data sent. We may also be constrained by the congestion window on the return path; if we include the data ACK in every ACK we will lose out to regular TCP. |
| Add/Remove IP | need to add sequence numbers and ACKs | need to add sequence numbers and ACKs |
| Add IP | can be sent on SYN | must be sent on subsequent packets |
| Subflow Priority | needs ordering and ACKs, as above | needs ordering and ACKs, as above |
| Security Negotiation | Too big to fit in options: must resort to payload encoding; can use option to indicate where the control data is | Can be done naturally. |
| Data FIN | No problem to do this with options, uses 1B of data space - the byte occupied by the subflow FIN | Must include new type of mapping for Data FIN that occupies 1B of payload. |
| Option Space Limitations | Currently a problem for security negotiation, everything else fits OK | no issue |
| **Existing Middleboxes** | | |
| Proactive ACKing | must use data ACK | must use data ACK |
| Stripping options on SYN | will not use path for secondary subflows, and will revert to TCP if first subflow | will not use path for secondary subflows, and will revert to TCP if first subflow |
| Stripping options on data segments | will not use path | will use path |
| **Multipath Middleboxes** | | |
| Stateless | can infer multipath and see flow of data | cannot distinguish between multipath and regular TCP |
| Stateful | can infer multipath, etc | can infer multipath if they see SYN handshake. If flows are rerouted, or middleboxes fail - can't distinguish from regular TCP. |
| **Implementation** | | |
| Data send | Must set data sequence mapping option on outgoing packets, reduce MSS, cut SACK if necessary | The connection level code simply embeds the mapping in the payload and sends the segment to the subflow |
| Data receive | Must signal data sequence mapping to connection level; possibly by swapping the sequence numbers in the segment after it is "in order" at subflow level. | Just passes payload to the connection level, which parses it to figure out what's needed. It needs to remove the control stuff from the segment before passing it up to the application. |
| Other options | same story as above; must be encoded on send as options (i.e. change subflow stack), must be signalled on receive (i.e. change subflow stack) | subflow stack unchanged, connection stack more complicated. |