Network working group
Internet-Draft
Expires: November 7, 2009

I. van Beijnum
IMDEA Networks
May 6, 2009

One-ended multipath TCP
draft-van-beijnum-1e-mp-tcp-00

Status of this Memo

Copyright Notice

Abstract

Normal TCP/IP operation is for the routing system to select a best path that remains stable for some time, and for TCP to adjust to the properties of this path to optimize throughput. A multipath TCP would be able to either use capacity on multiple paths, or

dynamically find the best performing path, and therefore reach higher throughput. By adapting to the properties of several paths through the usual congestion control algorithms, a multipath TCP shifts its traffic to less congested paths, leaving more capacity available for traffic that can't move to another path on more congested paths. And when a path fails, this can be detected and worked around by TCP much more quickly than by waiting for the routing system to repair the failure.

This memo specifies a multipath TCP that is implemented on the sending host only, without requiring modifications on the receiving host.

Table of Contents

1. Introduction

In order to achieve redundancy to protect against failures, network operators generally install more links than the minimum necessary to achieve reachability. So there are often multiple paths between any two given hosts, even when paths not allowed by policy are removed. However, routing protocols usually select a single "best" path. When multiple paths are used at the same time by the routing system, those tend to be parallel links between two routers or paths that are otherwise very similar. As such, a lot of potentially usable network capacity is left unused. A multipath transport protocol would be able to use more of that capacity by sending its data along multiple paths at the same time, or by switching to a path with more available capacity.

As TCP [RFC0793] is used by the vast majority of all networked applications, and TCP is responsible for the vast majority of all data transmitted over the internet, the logical choice would be to make TCP capable of using multiple paths. SCTP already has the ability to use multiple paths through the use of multiple addresses. However, using SCTP in this way requires significant application changes and deployment would be challenging because there is no obvious way for an application to know whether a service is available over SCTP rather than, or in addition to, TCP. In addition, SCTP as defined today [RFC2960] does not accommodate the concurrent use of multiple paths. Additional paths are purely used for backup purposes.

This memo describes a one-ended multipath TCP, which only changes the behavior of the TCP sender, achieving multipath advantages when communicating with unmodified TCP receivers. This means it is not possible to perform path selection by using different destination addresses. However, other mechanisms that are transparent to the receiver are possible. A simple one would be for the sender to send some packets to one router, and other packets to another router. If these routers then make different routing decisions for the destination address in the TCP packets, the packets flow over different paths part of the way. Other mechanisms to achieve the same goal are also possible. However, with a single destination address, paths can't be completely disjoint.

Using multiple paths at the same time brings up a number of challenges and questions:

- Naive scheduling (such as round robin) of transmissions over the different paths reduces performance of each path to that of the slowest path.

- Using multiple paths causes reordering, which triggers the fast retransmit algorithm, causing unnecessary retransmissions and reduced performance.

- TCP requires in-order delivery of data to the application, so when losses occur on one path, buffer capacity may run out and data can't be transmitted on unaffected paths until the lost data has been retransmitted.

- Using multiple paths with an instance of regular congestion control on each path for a single TCP session makes that session use network capacity more aggressively than single path sessions, which can be considered "unfair" and increases packet loss.

This memo seeks to address the first two issues by running separate instances of TCP's congestion control algorithms for the subflows that flow over different paths. Buffer issues are addressed by retransmitting packets before buffer space runs out, even if normal retransmission timers haven't fired yet. The fairness issue is a topic of ongoing research; this specification simply limits the number of subflows to limit unfairness and increased loss.

The one-ended multipath TCP takes advantage of the fact that TCP [RFC0793] congestion control [RFC2581] and flow control are performed by the sender. With regard to flow control and congestion control, the role of the receiver is limited to sending back acknowledgments and advertise how much data it is prepared to receive. Hence, it is possible for the sender to utilize different paths and modify the fast retransmit logic as long as the receiver recognizes the packets as belonging to the same session. So a multipath TCP sender can distribute packets over multiple paths as long as this doesn't require incompatible modifications to the IP or TCP header contents, most notably the addresses. A single-ended multipath TCP session must still be between a single source address and a single destination address, regardless of the path taken by packets.

The subset of the packets belonging to a TCP session flowing over a given path is designated a subflow.

In order to benefit from using multiple paths, it's necessary for the multipath TCP sender to execute separate TCP congestion control instances for the packets belonging to different subflows. In the case where all packets are subject to the same congestion window, performance over a fast and a slow path will often be poorer than over just the fast path, defeating the purpose of using multiple paths. For instance, in the case of a 10 Mbps and a 100 Mbps path with otherwise identical properties, a simple round robin distribution of the packets and the use of a single congestion window

will limit performance to that of the slowest path multiplied by the number of paths, 20 Mbps in this case.

2. Notational Conventions

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 [RFC2119].

3. Congestion control

A multipath TCP maintains instances of all congestion control related variables for each subflow. This includes, but is not limited to, the congestion window, the ssthresh, the retransmission timeout (RTO), the user timeout and RTT measurements. However, because TCP requires in-order delivery of data, there must be a single send buffer and a single receive buffer, thus flow control must happen session-wide.

Per-subflow congestion control is performed by recording the path used to transmit each packet. Acknowledgments are then attributed to the subflow the acknowledged packets were sent over and the congestion window and other congestion control variables for the relevant subflow are updated accordingly.

3.1. RTT measurements

Because a multipath TCP sender knows which packet it sent over which path, it can perform per-path round trip time measurements. This only works if return packets are consistently sent over the same path (or a set of paths with the same latency). If the receiver is not multipath-aware, this condition will generally hold: acknowledgments will flow from the receiver to the sender over a single path unless there is a topology change in the routing system or packets that belong to a single session are distributed over different paths by routers, which is rare. To multipath-capable routers on the return path (if any), the non-multipath-aware host appears to select the default path for all of its packets.

However, if, like the sender, the receiver is multipath-aware, then the return path that the receiver chooses to send ACKs over will influence the RTTs seen by the original sender. The situation where the sender is unaware of fact that the receiver selects different return paths with different latencies is suboptimal, even compared to consistently measuring the RTT over the slowest path, as this leads to higher variability in the RTT measurements and therefore a higher

RTO.

Having the receiver send ACKs over the same path mitigates the problem somewhat; but presumably, if the receiver is also multipath capable and has data to send, it will want to send this data over more than one path. So RTT measurements may inadvertently end up measuring different return paths in that case. A better solution is for the sender to include an indication in packets that allows the receiver to determine through which path the sender sent the packet. This information, along with the path initially chosen for the outgoing packet that is acknowledged, allows TCP to attribute each RTT measurement to a specific path.

Because congestion control happens per path, there must also be a separate retransmission timeout (RTO) value for each path.

3.2. Fast retransmit

Different paths will almost certainly have different RTTs, and even if the average RTT is the same, normal burstiness and differences in packet sizes will make packets routinely arrive through the different paths in a different order than the order in which they were transmitted. Without modifications to the algorithm, this would trigger the fast retransmit algorithm unnecessarily. To avoid this, fast retransmit is executed whenever, for packets belonging to the same subflow, after an unACKed packet or sequence of packets, more than two segments of new data is ACKed with SACK. This means fast retransmit happens per subflow, and reordering between subflows no longer triggers fast retransmit.

3.3. Slow retransmit

In multipath TCP, a per-path RTO is employed to recover from congestion events that fast retransmit can't handle. Because the missing packets create holes in the data stream, subsequent packets received over other paths must be buffered in the receive buffer. Unless the receive buffer is extremely large, this means the entire session stalls when the receive buffer fills up. This situation persists until the RTO expires for the congested or broken path so the missing packets can be retransmitted. Should the path in question be completely broken, this will then lead to an almost immediate new stall, and the stall/RTO cycles will then continue until the user timeout / R2 timer [RFC1122] for the subflow expires.

This is solved by taking unacknowledged packets transmitted over subflows that are stalled because they have exhausted their congestion window and are now waiting for the RTO to expire, and scheduling retransmissions of those packets over other paths before

the RTO of the stalled subflow expires. This should be done such that the missing packet arrives before it becomes necessary to stop sending data altogether because the receiver advertises a zero receive buffer. Such retransmissions therefore happen as the receive buffer space advertised by the receiver reaches RTT * MSS for the path that will be used for the retransmission; presumably the path with the lowest RTT. In essence, this creates a second level of fast retransmit that acts across subflows in addition to the normal fast retransmit that happens per subflow. This mechanism is named "slow retransmit".

In the case of single path TCP, scheduling retransmissions before the RTO expires could be problematic because this would be more aggressive than standard (New)Reno congestion control. But in the case of multipath TCP, the retransmission can happen over one of the other paths, which is still progressing.

By scheduling a retransmission faster than an RTO, there is an increased risk that a packet that was still working its way through the network is retransmitted unnecessarily. However, the alternative is allowing the progress of the session to stall (on all paths), reducing throughput significantly.

3.4. SACK

When packets (belonging to different subflows) arrive out of order, the the receiver can't acknowledge the receipt of the out of order packets using TCP's normal cumulative acknowledgment. However, the [RFC2018] (also see [RFC1072]) Selective Acknowledgment (SACK) mechanism is widely implemented. SACK makes it possible for a receiver to indicate that three or four additional ranges of data were received in addition to what is acknowledged using a normal cumulative ACK. When packets are sent over multiple paths and arrive out of order, the information in the SACK returned by the receiver can tell the sender how each subflow is progressing, so per-subflow congestion control can progress smoothly and unnecessary retransmissions are largely avoided.

One-ended multipath TCP requires the use of SACK to be able to determine which subflows are progressing even if other subflows are stalled, and thus the normal TCP ACK isn't progressing. If the remote host doesn't indicate the SACK capability during the three-way handshake, a multipath TCP implementation SHOULD limit itself to using only a single subflow and thus disabling multipath processing for the session in question.

3.5. Fairness and TCP friendliness

One of the goals of multipath TCP is increased performance over regular TCP. However, it would be harmful to realize this benefit by taking more than a "fair" share of the available bandwidth. One choice would be to make each subflow execute normal NewReno congestion control on each subflow, so that each individual subflow competes with other TCPs on the same footing as a regular TCP session. If all subflows use non-overlapping physical paths, other TCPs are no worse off than in the situation where the multipath TCP were a regular TCP sharing their path, so this could be considered fair even though the multipath TCP increases its bandwidth in direct relationship to the number of subflows used. Note that in this case, although multipath TCP sends at the same rate as regular TCP on a given path, resource pooling [wischik08pooling] benefits are still realized because a given transmission completes faster so it uses up resources for a shorter amount of time.

But if several logical paths share a physical path, multipath TCP takes a larger share of the bandwidth on that path. This would only be acceptable as fair for a very small number of subflows. The other end of the spectrum would be for multipath TCP to conform to exactly the same congestion window increase and decrease envelope that a regular TCP exhibits, being no more aggressive than a regular single path TCP session. At this point in time we will assume that fairness is a tunable factor of the regular NewReno AIMD envelope. A simple way to limit the amount of additional aggressiveness exhibited by multipath TCP is a limit on the number of subflows. Until more analysis has been performed and/or there is more experience with multipath TCP, a multipath TCP implementation SHOULD limit itself to using no more than 3 subflows concurrently.

4. Path selection

Note that in order to gain multipath benefits, the multipath TCP layer must be able to determine the logical path followed by each packet so it can measure path properties and perform per-path congestion control. In order to limit the number of packets flowing over each path to the amount allowed by the per path congestion window, the multipath TCP layer must be able to specify over which path a given packet is transmitted.

The situation where routers distribute packets over different paths based on their own criteria makes it impossible for hosts to send less traffic over congested paths and more traffic over uncongested paths and is therefore incompatible with multipath TCP. When routers distribute traffic belonging to the same flow (or, in the case of

multipath TCP: subflow) over different paths this will also cause reordering and the associated performance impact on TCP.

4.1. The multipath IP layer

The one-ended multipath TCP is logically layered on a multipath IP layer, which is able to to deliver packets to the same destination address through one or more logical paths, where the set of n logical paths share between one and m physical paths. In some cases, the multipath IP layer will be able to determine that a logical path isn't working, or maps to the same physical path as a previous logical path. For example, if the multipath TCP indicates that a packet should be sent over the third path, and the multipath IP is set up to use different next hop addresses for path selection, but only two next hop addresses are available, the multipath IP layer can provide feedback to the multipath TCP layer. In other cases, packets simply won't be delivered, or will be delivered through the same physical path used by other logical paths. This may for instance happen when multipath TCP selects path 1 and multipath IP puts a path selector with value "1" in the packet, but there are no multipath capable routers between the source and destination, so all packets, regardless of the presence and/or value of a path selector, are routed over the same physical path.

It is up to the multipath TCP layer to handle each of these situations.

For the purposes of this multipath TCP specification, the simplest possible interface to the multipath IP layer is assumed. When TCP segments traveling down the stack from the TCP layer to the IP layer aren't accompanied by a path selector value, or the path selector value is zero, the IP layer delivers packets in the same way as for unmodified TCP and other existing transport protocols, i.e., over the default path. Segments may also be accompanied by a path selector value higher than zero, which indicates the desired path. If the desired logical path is available, or may be available, the multipath IP layer attempts to deliver the packet using that logical path. If the desired logical path is known to be unavailable, the multipath IP layer drops the segment.

It is assumed that paths as seen by the multipath IP layer are mapped to logical paths with increasing numbers roughly ordered in order of decreasing assumed performance or availability. I.e., if path x doesn't work or has low performance, that doesn't necessarily mean that path x+1 doesn't work or has low performance, but if if paths x, x+1 and x+2 don't work or have low performance, then it's highly likely that paths x+3 and beyond also don't work or have even lower performance. Routers may have good next hop or even intra-domain

link weight information and link congestion information, but they generally don't have information about the end-to-end path properties, so the ordering of paths from high to low availability/ performance must be considered little more than a hint.

The multipath IP layer may be implemented through a variety of mechanisms, including but not limited to:

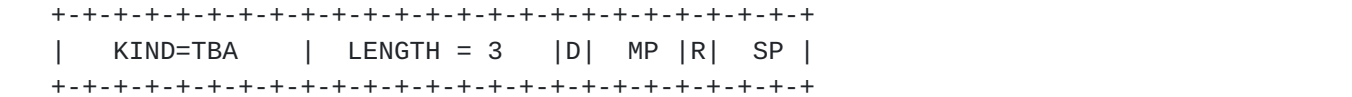
- Using different outgoing interfaces on the host
- Directing packets towards different next hop routers
- Integration with shim6 [I-D.ietf-shim6-proto] so that packets can use different address pairs
- Manipulation of fields used in ECMP [RFC2992] (i.e., a different flow label)
- Type of service routing (such as [RFC4915])
- Different lower layer encapsulation, such as MPLS
- Tunneling through overlays
- Source routing
- An explicit path selector field in packets, acted upon by routers

At this time, no choice is made between these different mechanisms.

4.2. The path indication option

Note that several of the fields discussed below are defined with future developments in mind, they are not necessarily immediately useful.

In order to allow for accurate RTT measurements and to inform the IP layer of the selected path, a TCP option indicating the desired path is included in all segments that don't use the default path. The format of this option is as follows:



The length is 3.

D is the "discard eligibility" flag (1 bit). It is similar, but not

identical, to the frame relay discard eligibility bit or the ATM cell loss priority bit. Set to zero, no special behavior is requested. Set to one, this indicates that loss of the packet will be inconsequential. This allows routers to drop packets with D=1 more readily than other packets under congested conditions, and also to completely block packets with D=1 on links that are considered long-term congested or expensive, even if there is no momentary congestion.

Setting the D bit to 1 for some subflows (presumably, ones with a performance lower than the best performing subflow) allows multipath TCP to give way to regular TCP and other single path traffic on congested or expensive paths. As long as the multipath TCP sets D to 0 on the subflow with the best performance, multipath TCP should still perform better than regular TCP, but the reduction in bandwidth use on the other paths helps achieve resource pooling benefits.

MP is a is a path selector that may be interpreted by multiple routers along the way (3 bits). A value of 0 is the default path that is also taken by packets that don't contain a multipath option. Multipath TCP aware routers should take this value into account when performing ECMP [RFC2992]. Packets with any value for MP MUST be forwarded, even if the number of available paths is smaller than the value in MP.

R (1 bit) is reserved for future use. MUST be set to zero on transmission and ignored on reception.

SP is a path selector that is interpreted only once by the local TCP stack or a router close to the sender (3 bits). A value of 0 is the default path that is also taken by packets that don't contain a multipath option. If the value in SP points to a path that isn't available, the packet SHOULD be silently dropped. This behavior, as opposed to selecting an alternate path out of the available ones, helps avoid the use of duplicate paths. As such, a router may only interpret SP rather than MP when it is known that the router is the only one acting on SP. All other routers may only act on MP.

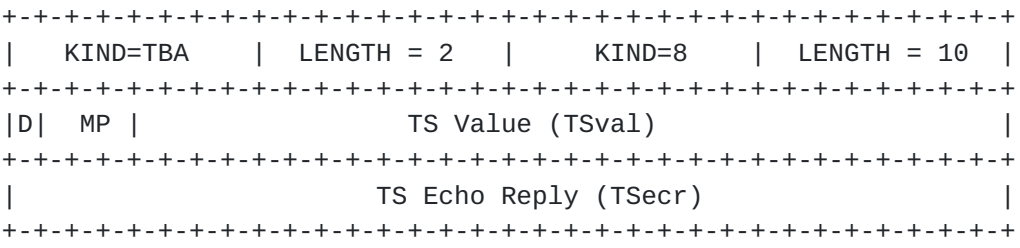
It is not expected that routers will make routing decisions directly based on the path indication option, as this option occurs deep inside the packet and not in a fixed place. However, a multipath IP layer or a middlebox may write a path selection value into a field in packets that is easily accessible to routers. But conceptually, the routers act upon the values in SP and MP.

The initial packets for each TCP session MUST use D, MP and SP values of zero. If D, MP and SP are all zero, then the path selector option isn't included in the packet. This makes sure that single path

operation remains possible even if packets with the path selector option are filtered in the network or rejected by the receiver. The packets that are part of the TCP three-way handshake SHOULD be sent over the default path, in which case they don't contain the path selector option; hence the ability to do multipath TCP isn't indicated to the correspondent at the beginning of the session as is usual for most other TCP extensions.

4.3. Timestamp integration option

As an optimization, hosts MAY borrow the four bits used by the path selector option from the timestamp option, and thus save one byte of option space, which means the path selector option can replace the padding necessary when the timestamp option is used and not increase header overhead. In that case, the combined path selector and timestamp options MUST appear as follows:



D and MP are the same as in the three-byte form of the path selector option. R and SP do not occur in this form of the path selector option and are assumed to be zero.

TSval is the locally generated timestamp. Because the timestamp is reduced to 28 bits, the minimum clock frequency is increased from the 59 nanoseconds mandated by [RFC1323] to 1 microsecond so the timestamp wraps in no less than 255 seconds.

TSecr is the timestamp echoed back to the other side (32 bits).

All hosts conforming to this specification MUST be able to recognize the integrated path selector and timestamp options, but they are not required to generate them.

4.4. Path for retransmissions

A multipath TCP implementation MUST be capable of scheduling retransmissions over a path different from the path used to transmit the packet originally. This includes packets subject to fast retransmit.

4.5. ECN

Explicit Congestion Notification works by routers setting a congestion indication in the IP header of packets rather than dropping those packets when they experience congestion. The receiver echos this information back to the sender which then performs congestion control in exactly the same way as if a packet was lost. The ECN specification ([RFC3168]) is such that the receiver sets the ECN-Echo (ECE) flag in the TCP header for all subsequent packets that it sends back until the sender sets the Congestion Window Reduced (CWR) flag. As the ECE flag is set in multiple ACKs, there is no obvious way to correlate the ECN indication in an ACK with a specific packet that experienced congestion, and subsequently, the path that is congested.

At this time, a multipath TCP conforming to this specification SHOULD NOT use ECN. ECN MAY be negotiated, but when more than a single path is used at a given time, packets SHOULD be sent with the ECN field set to Not-ECN (00), and incoming non-zero ECE flags SHOULD NOT be acted upon with regard to congestion control.

4.6. Path MTU discovery

Path MTU discovery [RFC1191] is performed for TCP by having TCP reduce its packet sizes whenever "packet too big but DF set" ICMP messages are received. As the name suggests, the path MTU is dependent on the path used, so multipath TCP must maintain MTU information for each path, and adjust this information for each path individually based on the too big messages that it receives.

The time between probing with a larger than previously discovered MTU must either be randomized or explicitly coordinated to avoid probing larger MTUs for multiple subflows at the same time, as probing larger MTUs is likely to lead to a lost packet, and having losses on multiple paths at the same time would be suboptimal. For instance, rather than probe every t, in the case of 2 paths, after t*0.5 the first path is probed, after t the second and after t*1.5 the first is probed again.

Both the IPv4 and IPv6 versions of ICMP return enough of the original packet in a "packet too big" message to be able to recover the sequence number from the original packet, which makes it possible to correlate the too big message with the packet that caused it, and thus the path used to transmit the packet.

5. Flow control and buffer sizes

In order to accommodate the increased number of packets in flight, the send buffer must be increased in direct relationship with the number of paths being used. Alternatively, the number of paths used concurrently should be limited to send buffer / avgRTT.

Although under normal operation, the receive buffer doesn't fill up, there are two reasons the receive buffer must be the same size as the send buffer: it must be able to accommodate a round trip time plus two segments worth of data during fast retransmit, and the advertised receive window limits the amount of data the sender will transmit before waiting for acknowledgments. So in practice, the receive buffer limits the maximum size of the send buffer, and therefore, the number of paths that can be supported concurrently.

There is no simple rule of thumb to determine the number of paths that should be used, as the maximum number of paths that the receive window can accommodate depends both on the maximum receive window advertised by the receiver and by the RTTs on the paths.

6. Handling of RSTs

If an RST is received after enabling a new path, this could be a reaction to the presence of an unknown option. So the optimal situation would be for an RST to reset just the path used to send the packet that generated the RST, not the entire session. Only when the last path or the default path (on which packets don't include special options) receives an RST, the entire session should be reset.

7. Middlebox considerations

NATs are designed to be transparent to TCP. Because one-ended multipath TCP conforms to normal TCP semantics on the wire, multipath TCP should in principle also be compatible with NAT. However, if different paths are served by different NATs that apply different translations, the receiver won't be able to determine that the different subflows through the different paths belong to the same TCP session. So for NAT to work, the translation must either happen in a location that all paths flow through, or the different NATs on the different paths must act as a single, distributed NAT and apply the same translation to the different subflows.

Middleboxes that only see traffic flowing over a subset of the paths used will see large numbers of gaps in the sequence number space. They may also not observe only a partial three-way handshake, or not

observe any ACKs. As such, like with NATs, middleboxes that enforce conformance to known TCP behavior, must be placed such that they observe all subflows. For middleboxes that just check whether packets fall inside the TCP window, it may be sufficient for multipath TCP senders to make sure that all paths see at least one packet per window. Middleboxes that enforce sequence number integrity will almost certainly also block TCP packets for which they didn't observe the three way handshake. A possible way to accommodate that behavior would be to send copies of all session establishment and tear down packets over all paths that the sender may use. However, this strategy is still likely to fail unless the receiver does the same so the middleboxes may observe the signaling packets flowing in both directions.

It's also possible that middleboxes (or perhaps hosts themselves) reject packets with the path indicator TCP option. Since packets flowing over the default path don't carry the path indicato option, these packets should always be allowed through, so single path operation is always possible. When a multipath TCP sender starts to send packets over alternative paths, those packets won't make it to the receiver because they contain the path indicator option. The result is that a new subflow, which would use a congestion window of two maximum segment sizes, would send two packets and then experiences a retransmission timeout. Slow retransmit makes sure the packets are transmitted before the session stalls, so the impact of the lost packets is negligible.

8. Security considerations

None at this time.

9. IANA considerations

IANA is requested to provide a TCP option kind number for the path indication option.

10. Acknowledgements

The single ended multipath TCP was developed together with Marcelo Bagnulo and Arturo Azcorra.

Members of the Trilogy project, especially Costin Raiciu, have contributed valuable insights.

Iljitsch van Beijnum 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

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, November 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

11.2. Informational References

[RFC1072] Jacobson, V. and R. Braden, "TCP extensions for long-delay paths", RFC 1072, October 1988.

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

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, June 2007.

[wischik08pooling] Wischik, D., Handley, M., and M. Bagnulo Braun, "The resource pooling principle", Computer Communication Review 38, September 2008.

[I-D.ietf-shim6-proto] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", draft-ietf-shim6-proto-11 (work in progress), December 2008.

Appendix A. Document and discussion information

The latest version of this document will always be available at http://www.muada.com/drafts/. Please direct questions and comments to the multipathtcp@ietf.org mailinglist or directly to the author.

Appendix B. An implementation strategy

In order to perform per-path congestion control, all of the ACK-based events that trigger congestion control responses as well as all the variables used by the congestion control algorightms must be recreated in the multipath situation. These are the triggers and variables for the four mechanisms in RFC 2581.

1. the path MTU (page 4)

2. the arrival of an ACK that acknowledges new data (page 4)

3. the arrival of a non-duplicate ACK (page 4) or the sum of new data acknowledged (page 5)

4. triggering of the retransmission timer (page 5)

5. the flightsize or number of bytes sent but not acknowledged (page 5)

6. the retransmission of a segment (page 5)

7. the arrival of a third or subsequent duplicate ACK (page 6, page 7)

8. whether a retransmission timeout period has elapsed since the last reception of an ACK (page 7)

1, 4, 6 and 8 are maintained session-wide.

We recreate these events and variables based on SACK information in the one-sequence number multipath TCP case as follows.

We keep track of every packet sent. (Alternatively: multi-packet contiguous blocks of data transmitted over the same path.) When an ACK comes in, we first remove the stored information about packets/ data blocks that are cumulatively ACKed, noting how much data was ACKed for each path that the packets were sent over. Then we do the same for all the SACK blocks in the ACK. Because we remove the information about (S)ACKed data and you can remove something just once, we don't have to keep track of previous SACKs like the current BSD implementation does.

The only slightly tricky part is emulating duplicate ACKs. This may not even be really necessary, as the SACKs give us better information to base fast retransmit on, but that's something for another day. What happens in the pseudo code is that when traversing the list of sent packets (this happens in order of seqnum), we note the path that packets that aren't SACKed are sent over. When we're done processing SACK data and it turns out that for a path there are one or more packets that we skipped over when processing SACK data and there was also data SACKed after a skipped packet, there was a lost (or reordered) packet on this path. When the amount of "duplicate ACKed" data grows beyond two segment sizes, we've reached the equivalent of three duplicate ACKs so we trigger fast retransmit (7).

We update the congestion window (2 and 3) when there was data (S)ACKed for a path. ACKs that don't acknowledge any data for a path aren't relevant because we don't need them to trigger fast retransmit and we assume that they're sent to (S)ACK data for other paths, anyway. (Or they could be window updates.)

We maintain the flightsize (5) by simply adding data bytes as packets are transmitted and subtracting when they're (S)ACKed. Because we have explicit SACKs, we don't need to guess based on duplicate ACKs. The flightsize is also adjusted when we perform fast retransmit or a regular retransmission over a path other than which was used for the original packet. In addition, we explicitly mark some packets to trigger once-per-RTT actions when they're ACKed.

Pseudo code for the above:

```
   // initializing data structures is left as an exercise for the
   // reader

   // transmitting packets
   // assume we've selected a path to transmit over

   path.flightsize = path.flightsize + packet.datasize
   packet.path = path
   packet.status.acked = false
   // set up state to remember to do per RTT stuff when packet is
   // ACKed
   if path.do_per_rtt_next_packet == true
     path.per_rtt_seqnum = packet.seqnum.first
     packet.per_rtt = true
     path.do_per_rtt_next_packet = false
   else
     packet.status.per_rtt = false
   // don't set ECN on outgoing packets for now, can add logic
   // for deciding which packets to ECN enable later
   packet.ecn.sent = 0
   // add to linked list of sent packets (to handle retrans-
   // missions, linked list must maintain seqnum order, not FIFO
   // or LIFO)
   llpush(packet)

   // receiving (S)ACKs

   // normal flow-wide flow control actions based on cumACK
   // also happen (elsewhere)

   // handle ECN, must detect transitions rather than
   // depend on actual value
   if packet.ecnecho == true
     if ecn.previous == true
       ecn.current = false
     else
       ecn.current = true
       ecn.previous = true
   else
     ecn.previous = false

   // initialize some stuff before we handle the ACK
   for each path
     path.do_per_rtt = false
     path.ackedbytes = 0
     path.unacked.sure = 0
     path.unacked.maybe = 0
     path.ecn.received = false
```

```
   // remove cumulatively ACKed packets
   llwalk_init
   packet = llwalk_next
   while packet.seqnum.first < ack.cumulative
     // ECN, we only act if we enabled ECN when we sent the packet
     if ecn.current & packet.ecn.sent <> 0
       path.ecn.received = true
     // if part of a packet is ACKed, we need some trickery
     if packet.seqnum.last_plus_one > ack.cumulative
       path.ackedbytes += ack.cumulative - packet.seqnum.first
       packet.seqnum.first = ack.cumulative
     else
       path.ackedbytes = path.ackedbytes + packet.datasize
       if packet.per_rtt & packet.seqnum.first == path.per_rtt_seqnum
         path.do_per_rtt = true
       llremove(packet)
     packet = llwalk_next

   // now we handle the SACKs (assume exactly one SACKblock for
   // simplicity) we continue walking the linked list, no need to
   // restart
   while packet.seqnum.first < ack.sack.last_plus_one
     if packet.seqnum.last_plus_one < ack.sack.first
       // these packets overlap with the SACK block
       // for simplicity, assume packets are always completely
       // SACKed in reality we need to split a packet if only the
       // middle is SACKed ECN, we only act if we enabled ECN when
       // we sent the packet
       if ecn.current & packet.ecn.sent <> 0
         path.ecn.received = true
       path.ackedbytes = path.ackedbytes + packet.datasize
       if packet.per_rtt & packet.seqnum.first == path.per_rtt_seqnum
         path.do_per_rtt = true
       // add potentially unacked bytes to for sure unacked bytes
       // because we now know we had a SACK hole if any
       // unacked maybe bytes
       path.unacked.sure = path.unacked.sure + path.unacked.maybe
       path.unacked.maybe = 0
       // remove packet from the list
       llremove(packet)
     else
       // note how many bytes we skipped unSACKed
       // if later data is SACKed, that's our version of a dup ACK
       path.unacked.maybe = path.unacked.maybe + packet.datasize
     packet = llwalk_next

   // done processing, now tally up the the results
   foreach path
```

```
    // update flightsize (item 5 in CC events/variables list)
    path.flightsize = path.flightsize - path.ackedbytes
    // if any data was ACKed
    if path.ackedbytes <> 0
      // some stuff was ACKed for this path
      if path.unacked.sure > 2 * path.mss
        // more than 2 * MSS worth of data in SACK hole = fast
        // retransmit execute fast retransmit (item 7 in CC
        // events/variables list) need to handle flightsize in
        // some way here ignore ECN because we already have a loss
        // send back ECN window update indication, though
      else
        // SACKs were cumulative for this path
        // execute cwnd update (items 2 and 3 in CC events/
        // variables list)
        // ECN must be taken into account here
        // and send back ECN window update indication
      if path.do_per_rtt
        // execute per RTT actions
        // indicate that this should be set for next packet sent
        path.do_per_rtt_next_packet == true
```

Note that the pseudo-code doesn't cover all the mechanisms explained earlier. Also, ECN is handled here because it's not too difficult to do. The hard part is deciding which packets to enable ECN for.

Author's Address

Iljitsch van Beijnum
IMDEA Networks
Avda. del Mar Mediterraneo, 22
Leganes, Madrid 28918
Spain

Email: iljitsch@muada.com