MPTCP Working Group
Internet-Draft
Intended status: Experimental
Expires: January 4, 2018

F. Duchene
UCLouvain
V. Olteanu
University Politehnica of Bucharest
O. Bonaventure
UCLouvain
C. Raiciu
University Politehnica of Bucharest
A. Ford
Pexip

July 03, 2017

Multipath TCP Load Balancing
draft-duchene-mptcp-load-balancing-01

Abstract

In this document we propose several solutions to allow Multipath TCP to better work behind load balancers.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Table of Contents

1. Introduction

Multipath TCP is an extension to TCP [RFC0793] that was specified in [RFC6824]. Multipath TCP allows hosts to use multiple paths to send and receive the data belonging to one connection. For this, a Multipath TCP connection is composed of several TCP connections that are called subflows.

Many large web sites are served by servers that are behind a load balancer. The load balancer receives the connection establishment attempts and forwards them to the actual servers that serve the requests. One issue for the end-to-end deployment of Multipath TCP is its ability to be used on load-balancers. Different types of load balancers are possible. We consider a simple but important load balancer that does not maintain any per-flow state. This load balancer is illustrated in Figure 1. A stateless load balancer can be implemented by hashing the five tuple (IP addresses and port numbers) of each incoming packet and forwarding them to one of the servers based on the hash value computed. With TCP, this load balancer ensures that all the packets that belong to one TCP connection are sent to the same server since each packet contains the five-tuple used by the hash function.

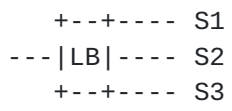

Figure 1: Stateless load balancer

With Multipath TCP, this approach cannot be used anymore when subflows are created by the clients. Such subflows can contain any five tuple and thus packets belonging to them will be load-balanced to any server, not necessarily the one that was selected by the hashing function for the initial subflow.

In this document, we propose several solutions to allow Multipath TCP to work behind load balancers.

2. Proposed solutions

2.1. Per-server addresses

A first solution is to use two types of public addresses. The load balancer uses a public address that is advertised in the DNS. This address is used to establish the initial subflow of all Multipath TCP connections. In addition to this address, a pool of addresses is used for the servers behind the load balancer. One address of this pool is assigned to each server behind the load balancer. This server address is not announced in the DNS and only advertised by the servers through the ADD_ADDR option.

The additional per-server address is used by the clients when they wish to create additional subflows. Since each server has its own public address, this ensures that the additional subflows are directed to the corresponding server. For this solution, we need to ensure that the client never use the public address of the load balancer to initiate subflows. This can be achieved by a slight modification to the MP_CAPABLE option described below.

To allow Multipath TCP to work for servers behind layer 4 load balancers, we propose to use the reserved "B" flag in the MP_CAPABLE option sent (shown in Figure 2 in the SYN+ACK. This flag informs the other host that this address MUST NOT be used to create additional subflows.

A host receiving an MP_CAPABLE with the "B" set to 1 MUST NOT try to establish a subflow to the source address of the MP_CAPABLE. This bit can also be used in the MP_CAPABLE option sent in the SYN by a client that resides behind a NAT or firewall or does not accept server-initiated subflows.

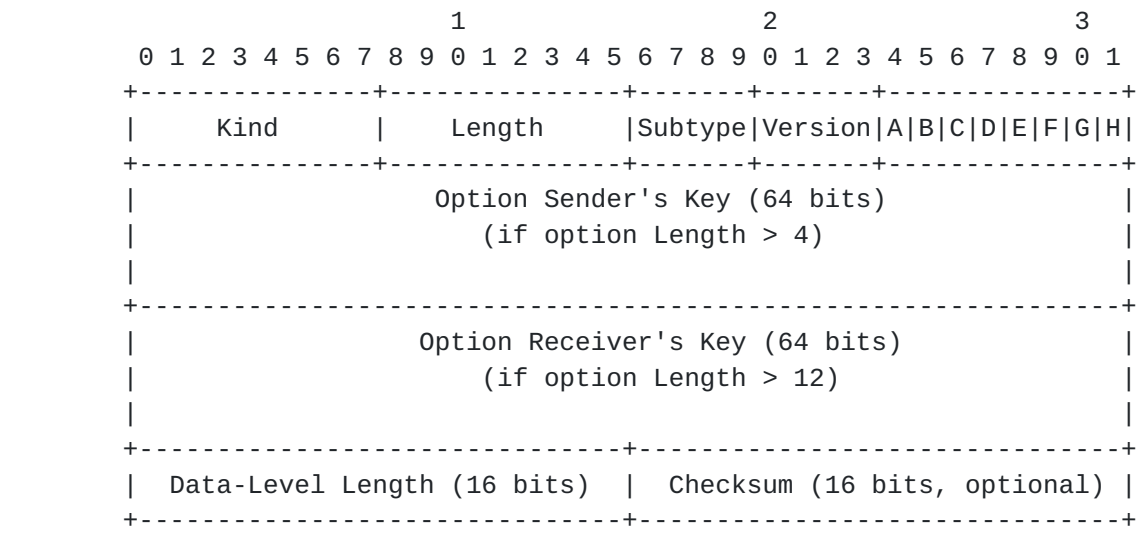

Figure 2: Multipath Capable (MP_CAPABLE) Option

This bit can be used by servers behind a stateless load balancer. The servers set the "B" flag in the MP_CAPABLE option that they return and advertise their own address by using the ADD_ADDR option. Upon reception of this option, the clients can create the additional subflows towards these addresses. Compared with current stateless load balancers, an advantage of this approach is that the packets belonging to the additional subflows do not need to pass through the load balancer.

To demonstrate the principle of an off path load balancer let's consider a server behind a load balancer.

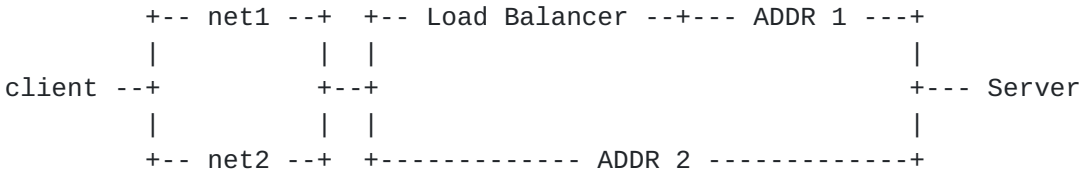

Figure 3: A server with 2 addresse.

As shown in figure Figure 3, this server has 2 IP addresses: 1 behind the load balancer and 1 directly connected to the Internet. The client sends a SYN containing an MP_CAPABLE option, the server answers with a SYN+ACK containing an MP_CAPABLE with the "B" flag set to 1. Upon reception of the SYN+ACK, the client will know that it cannot establish any more subflow towards IP address. The server will then advertise it's secondary address with an ADD_ADDR. Once the client has established at least one connection to the secondary

IP address, the server could elect to close the primary subflow or to put it in backup mode.

2.2. Embedding Extra Information in Packets

Under some circumstances, addressing the individial servers via their individial IPs is not desirable or feasible. To work around this issue, we propose two mutually-exclusive solutions. They rely to varying degrees on getting the client to embed connection or server-identifying information in the packets that it sends out. This extra information can be used statelessly by the loadbalancers.

Both solutions require modifications only to the server stack and work well with existing MPTCP clients.

2.2.1. Proposal 1

Our first proposal revolves around controlling the destination port that the client uses in all subflows aside from the initial one. It is possible for the server to advertise an additional port via the ADD_ADDR option [RFC6824]. This informs the client that it can send an MP_JOIN to this new port and initiate a new subflow.

To take advantage of this, each server is be assigned a unique 16-bit ID, which must be different from the port on which the service is being hosted (e.g. 80). As soon as a connection is initiated, the server sends an ADD_ADDR to the client advertising a new port equal to said ID.

Packets that arrive at the loadbalancer are treated as follows:

- Packets destined to the port that the service is being hosted on will be forwarded to a server based on a hash of the 5-tuple.
- Packets destined to any other port are forwarded to the server whose ID matches the destination port.

This approach has two drawbacks:

- The client will most likely also try to initiate subflows using the server's original port. Because these subflows are loadbalanced based on a hash of their 5-tuple, they will almost certainly reach a different server and break. (Using REMOVE_ADDR to prevent the creation of these subflows would entail the destruction of the original subflow.) This issue can be solved by the adoption of the protocol modifications outlined in Section 2.1.

- If the client is behind a firewall that restricts access to certain destination ports, it might not succeed in establishing any new subflows.

2.2.2. Proposal 2

Our second proposal is to loadbalance packets based on the server's token.

The token's most significant 14 bits are treated as a hash value for the connection. They are embedded in all outgoing TCP timestamps, and subsequently echoed back by the client. Incoming packets that do not contain timestamps (such as FINs) are dealt with via redirection between the servers.

2.2.2.1. Connection Initiation

The client initiates an MPTCP connection by sending a SYN with the MP_CAPABLE option. Under normal operation, the server then picks a random 64-bit key for the connection, and uses it to compute its token.

To forward the packet appropriately, the load balancer must know the token before deciding what server to send it to. To accomplish this, we move the key generation to the load balancer. The connection's token can be computed based on the generated key.

The load balancer places the generated key, along with the IP address of the server that would be responsible for the subflow under normal 5-tuple hashing (which we call the alternate server IP) in an IP option and forwards the SYN to the server.

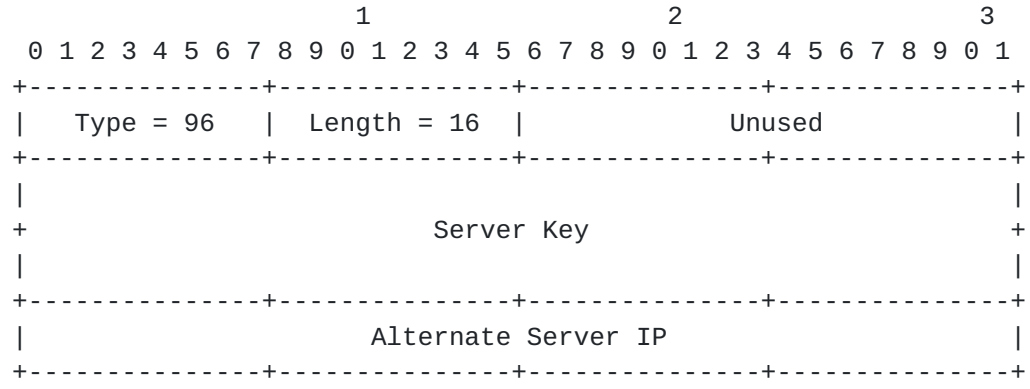

Figure 4: IP Option Used for MP_CAPABLE packets

The figure above depicts the IP option that is inserted into the MP_CAPABLE packet before it is sent to the server. We have chosen an IP option despite the fact that the data contained therein pertains to the transport layer, because TCP option space is very limited. IP option type 96 is currently classified as reserved [RFC0791].

Upon receipt of the packet, the server uses the key provided to compute the token for the connection. If no connection with the same token exists, the server uses the key provided. Otherwise, it takes a brute-force approach and randomly generates multiple keys and selects one that yields a token with the same 14 highest-order bits.

The use of the alternate server IP will be discussed in a later section.

2.2.2.2. Handling MP_JOIN packets

Additional subflows are initiated by the client by sending MP_JOIN packets. These packets contain the server's token.

Similarly to how MP_CAPABLE packets are treated, the load balancer uses an IP option to inform the server about which other server would be responsible for the subflow under normal 5-tuple hashing.

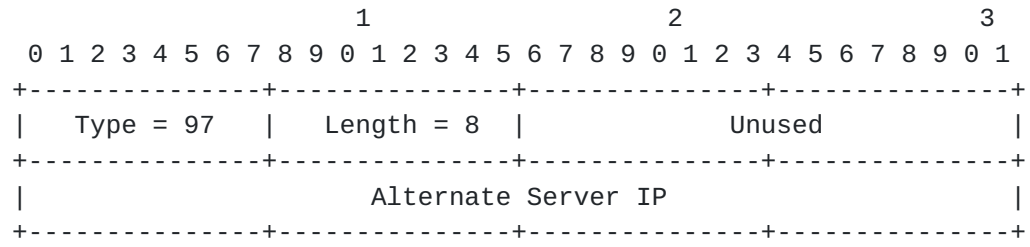

Figure 5: IP Option Used for MP_JOIN packets

IP option type 97 is also classified as reserved [RFC0791].

2.2.2.3. Embedding the token in the timestamp

The TCP timestamp option [RFC7323] is present in most packets and is comprised of two fields: the TSval, which is set by the packet's sender, and TSecr, which contains a timestamp recently received from the other end.

Taking advantage of the fact that timestamps set by the server are echoed back by the client, the server shifts its timestamp clock left by 14 bits, and embeds the 14 highest-order bits of the token into the 14 lowest-order bits of the TSval. When a packet with the ACK

flag set and with the TS option present arrives at the loadbalancer, it is forwarded based on the 14 least significant bits of the TSecr field.

2.2.2.3.1. Impact on PAWS

Timestamps supplied by the server are used by the client for protection against wrapped sequence numbers (PAWS). Note that for Multipath TCP, the utilisation of the 64 bits DSN already protects against PAWS.

We assume that the server uses a timestamp clock frequency of 1 tick per ms, which is the highest frequency recommended by [RFC7323]. The recycling time of the timestamp clock's sign bit is required to be greater than the Maximum Segment Lifetime of 255 seconds. Given that the clock ticks once every ms in increments of 2 ^ 14, its recycling time is roughly 262 s, which is within the bounds set by the standard.

While the quickly-increasing timestamp is benign to active subflows, PAWS will still cause segments to be dropped if the subflow in question had been idle for a period longer than the clock's recycling time. To solve this, the server periodically sends keepalive messages during idle periods.

2.2.2.4. Redirecting packets without timestamps

Some packets (most notably FINs) do not contain timestamps or any other connection-identifying information. As such, they are forwarded to a server based on a hash of the 5-tuple.

As seen in Section 2.2.2.1 and Section 2.2.2.2, whenever a new subflow is setup, the server responsible for it (A) also knows which other server (B) would be hit by the packets in case 5-tuple hashing is used.

A will use a simple peer-to-peer protocol to inform B to setup a redirection rule for the 5-tuple in question. The redirection rule will be deleted by B either at A's request, after the subflow has finished, or after a timeout. We do not discuss the specifics of the protocol in this document.

Redirection of a packet is performed using IP-in-IP encapsulation.

2.3. Application Layer Authentication

With similar motivations to 2.2, this proposal [I-D.paasch-mptcp-application-authentication] decouples the token signalled in the TCP options from the key used in authentication, allowing the token to carry arbitrary information. By allowing the token to be arbitrarily assigned by the sender, a load balancer could embed routing information so it knows which server to forward the packets on the TCP session towards.

For example, the token could carry a server identifier, a port number, and a signature based on a known secret. Furthermore, by generating tokens directly there is no risk of hash collisions in token generation. By allowing the token to be arbitrarily assigned, decoupled from the keys, the authentication of additional subflows is delegated to the application layer. A proposal for the use of TLS for this is defined in [I-D.paasch-mptcp-tls-authentication], whereby keys can be extracted from a TLS session and used to set up additional subflows.

3. Comparaison of the solutions

Per-server addresses:

- Requires individual public addresses for each of the servers, making IPv6 almost mandatory.
- Requires modifications to the clients and servers stack.
- Is transparent and works with today's load balancers.
- Doesn't need any modification to the applications.
- Disclose the real IP address of the servers.
- Allows to put the load balancer off-path.

Extra Information in Packets:

- Doesn't require an individual public addresses for each of the servers.
- Requires modifications to the load balancers servers stack.
- Could be broken by a firewall blocking certain destination ports (proposal 1) or changing the value of the timestamps (proposal 2).
- Doesn't need any modification to the applications.

- Doesn't disclose the real IP address of the servers.

Application Layer Authentication:

- Doesn't require public IP addresses
- Requires support at clients and load balancers
- Doesn't disclose IP addresses
- No greater risk of middle box interference than MPTCP today
- Additional security through no key exchange in the clear

4. Recommandations

5. IANA considerations

This document proposes some modifications to the Multipath TCP options defined in [RFC6824]. These modifications do not require any specific action from IANA.

6. Security considerations

Security considerations will be discussed in the next version of this draft.

7. Conclusion

In this document, we have described and compared two solutions to load balance MultiPath TCP connections. We showed that these two solutions have advantages and drawbacks and cover different network configurations. Future versions of this draft will discuss security considerations.

8. References

8.1. Normative References

[I-D.paasch-mptcp-application-authentication]
Paasch, C. and A. Ford, "Application Layer Authentication for MPTCP", draft-paasch-mptcp-application-authentication-00 (work in progress), May 2016.

[I-D.paasch-mptcp-tls-authentication]
Paasch, C. and A. Ford, "TLS Authentication for MPTCP", draft-paasch-mptcp-tls-authentication-00 (work in progress), May 2016.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <http://www.rfc-editor.org/info/rfc791>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <http://www.rfc-editor.org/info/rfc7323>.

8.2. Informative References

[I-D.ietf-mptcp-rfc6824bis] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-07 (work in progress), October 2016.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <http://www.rfc-editor.org/info/rfc1323>.

[RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, DOI 10.17487/RFC6182, March 2011, <http://www.rfc-editor.org/info/rfc6182>.

[RFC7430] Bagnulo, M., Paasch, C., Gont, F., Bonaventure, O., and C. Raiciu, "Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP)", RFC 7430, DOI 10.17487/RFC7430, July 2015, <http://www.rfc-editor.org/info/rfc7430>.

Authors' Addresses

Fabien Duchene
UCLouvain

Email: fabien.duchene@uclouvain.be

Vladimir Olteanu
University Politehnica of Bucharest

Email: vladimir.olteanu@cs.pub.ro

Olivier Bonaventure
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be

Costin Raiciu
University Politehnica of Bucharest

Email: costin.raiciu@cs.pub.ro

Alan Ford
Pexip

Email: alan.ford@gmail.com