MPTCP Working Group C. Paasch
Internet-Draft G. Greenway
Intended status: Experimental Apple, Inc.
Expires: March 10, 2016 A. Ford
Pexip
September 7, 2015

Multipath TCP behind Layer-4 loadbalancers
draft-paasch-mptcp-loadbalancer-00

Abstract

Large webserver farms consist of thousands of frontend proxies that serve as endpoints for the TCP and TLS connection and relay traffic to the (sometimes distant) backend servers. Load-balancing across those server is done by layer-4 loadbalancers that ensure that a TCP flow will always reach the same server.

Multipath TCP's use of multiple TCP subflows for the transmission of the data stream requires those loadbalancers to be aware of MPTCP to ensure that all subflows belonging to the same MPTCP connection reach the same frontend proxy. In this document we analyze the challenges related to this and suggest a simple modification to the generation of the MPTCP-token to overcome those challenges.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Problem statement 3
3. Proposals 4
3.1. Explicitly announcing the token 4
3.2. Changing the token generation 6
4. Conclusion 6
5. IANA Considerations 7
6. References 7
6.1. Normative References 7
6.2. Informative References 7
Authors' Addresses 7

1. Introduction

Internet services rely on large server farms to deliver content to the end-user. In order to cope with the load on those server farms they rely on a large, distributed load-balancing architecture at different layers. Backend servers are serving the content from within the data center to the frontend proxies. These frontend proxies are the ones terminating the TCP connections from the clients. A server farm relies on a large number of these frontend proxies to provide sufficient capacity. In order to balance the load on those frontend proxies, layer-4 loadbalancers are installed in front of these. Those loadbalancers ensure that a TCP-flow will always be routed to the same frontend proxy. For resilience and capacity reasons the data-center typically deploys multiple of these loadbalancers [Shuff13] [Patel13].

These layer-4 loadbalancers rely on consistent hashing algorithms to ensure that a TCP-flow is routed to the appropriate frontend proxy. The consistent hashing algorithm avoids state-synchronization across the loadbalancers, making sure that in case a TCP-flow gets routed to a different loadbalancer (e.g., due to a change in routing) the TCP-flow will still be sent to the appropriate frontend proxy [Greenberg13].

Multipath TCP uses different TCP flows and spreads the application's data stream across these [RFC6824]. These TCP subflows use a different 4-tuple in order to be routed on a different path on the Internet. However, legacy layer-4 loadbalancers are not aware that these different TCP flows actually belong to the same MPTCP connection.

The remainder of this document explains the issues that arise due to this and suggests a possible change to MPTCP's token-generation algorithm to overcome these issues.

2. Problem statement

In an architecture with a single layer-4 loadbalancer but multiple frontend proxies, the layer-4 loadbalancer will have to make sure that the different TCP subflows that belong to the same MPTCP connection are routed to the same frontend proxy. In order to achieve this, the loadbalancer has to be made "MPTCP-aware", tracking the keys exchanged in the MP_CAPABLE handshake. This state-tracking allows the loadbalancer to also calculate the token associated with the MPTCP-connection. The loadbalancer thus creates a mapping (token, frontend proxy), stored in memory for the lifetime of the MPTCP connection. As new TCP subflows are being created by the client, the token included in the SYN+MP_JOIN message allows the loadbalancer to ensure that this subflow is being routed to the appropriate frontend proxy.

However, as soon as the data center employs multiple of these layer-4 loadbalancers, it may happen that TCP subflows that belong to the same MPTCP connection are being routed to different loadbalancers. This implies that the loadbalancer needs to share the mapping-state it created for all MPTCP connections among all other loadbalancers to ensure that all loadbalancers route the subflows of an MPTCP connection to the same frontend proxy. This is substantially more complicated to implement, and would suffer from latency issues.

Another issue when MPTCP is being used in a large server farm is that the different frontend proxies may generate the same token for different MPTCP connections. This may happen because the token is a truncated hash of the key, and hash collisions may occur. A server farm handling millions of MPTCP connections has actually a very high chance of generating those token-collisions. A loadbalancer will thus no more be able to accurately send the SYN+MP_JOIN to the correct frontend proxy in case a token-collision happened for this MPTCP connection.

3. Proposals

The issues described in Section 2 have their origin due to the undeterministic nature in the token-generation. Indeed, if it becomes possible for the loadbalancer to infer the frontend proxy to forward this flow to, MPTCP becomes deployable in such kinds of environments.

The suggested solutions have their basis in a token from which a loadbalacer can glean routing information in a stateless manner. To allow the loadbalancer to infer the proxy based on the token, the proxies each need to be assigned to a range of unique integers. When the token falls within a certain range, the loadbalancer knows to which proxy to forward the sufblow. Using a contiguous range of integers makes the frontend very vulnerable to attackers. Thus, a reversible function is needed that makes the token random-looking. A 32-bit block-cipher (e.g., RC5) provides this random-looking reversible function. Thus, for both proposals we assume that the frontend proxies and the layer-4 loadbalancer share a local secret Y, of size 32 bits. This secret is only known to the server-side data center infrastructure. If X is an integer from within the range associated to the proxy, the proxy will generate the token by encypting X with secret Y. The loadbalancer will simply decrypt the token with the secret Y, which provides it the value of X, allowing it to forward the TCP flow to the appropriate proxy.

This approach also ensures that the tokens generated by different servers are unique to each server, eliminating the token-collision issue outlined in the previous section.

In the following we outline two different approaches to handle the above described problems, using this approach. The two proposals provide different ways of communicating the token over to the peer during the MP_CAPABLE handshake. We would like these proposals to serve as a discussion basis for the design of the definite solution.

3.1. Explicitly announcing the token

One way of communicating the token to simply announce it in plaintext within the MP_CAPABLE handshake. In order to allow this, the wire-format of the MP_CAPABLE handshake needs to change however.

One solution would be to simply increase the size of the MP_CAPABLE by 4 bytes, giving space for the token to be included in the SYN and SYN/ACK as well as adding it to the third ACK. However, due to the scarce TCP-option space this solution would suffer deployment difficulties.

If the solution proposed in [I-D.paasch-mptcp-syncookies] is being deployed, the MP_CAPABLE-option in the SYN-segment has been reduced to 4 bytes. This gives us space within the option-space of the SYN-segment that can be used. This allows the client to announce its token within the SYN-segment. To allow the server to announce its token in the SYN/ACK, without bumping the option-size up to 16 bytes, we reduce the size of the server's key down to 32 bits, which gives space for the server's token. To avoid introducing security-risks by reducing the size of the server's key, we suggest to bump the client's key up to 96 bits. This provides still a total of 128 bits of entropy for the HMAC computation. The suggested handshake is outlined in Figure 1.

```
    SYN + MP_CAPABLE_SYN (Token_A)
-------------------------------------->
  (the client announces the 4-byte locally
   unique token to the server in the
   SYN-segment).


   SYN/ACK + MP_CAPABLE_SYNACK (Token_B, Key_B)
<--------------------------------------
  (the server replies with a SYN/ACK announcing
   as well a 4-byte locally unique token and a 4-byte key)


   ACK + MP_CAPABLE_ACK (Key_A, Key_B)
--------------------------------------->
   (third ack, the client replies with a 12-byte Key_A
    and echoes the 4-byte Key_B as well).

The suggested handshake explicitly announces the token.
```

Figure 1

Reducing the size of the server's key down to 32 bits might be considered a security risk. However, one might argue that neither parties involved in the handshake (client and server) have an interest in compromising the connection. Thus, the server can have confidence that the client is going to generate a 96 bits key with sufficient entropy and thus the server can safely reduce its key-size down to 32 bits.

However, this would require the server to act statefully in the SYN exhcnage if it wanted to be able to open connections back to the client, since the token never appears again in the handshake.

3.2. Changing the token generation

Another suggestion is based on a less drastic change to the MP_CAPABLE handshake. We suggest to infer the token based on the key provided by the host. However, in contrast to [RFC6824], the token is not a truncated hash of the keys. The token-generation uses rather the following scheme: If we define Z as the 32 high-order bits and K the 32 low-order bits of the MPTCP-key generated by a host, we suggest to generate the token as the encryption of Z with key K by using a 32-bit block-cipher (the block-cipher may for example be RC5 - it remains to be defined by the working-group which is an appropriate block-cipher to use for this case). The size of the MPTCP-key remains unchanged and is actually the concatenation of Z with K. Both, K and Z are different for each and every connection, thus the MPTCP-key still provides 64 bits of randomness.

Using this approach, a frontend proxy can make sure that a loadbalancer can derive the identity of the backend server solely through the token in the SYN-segment of the MP_JOIN exchange, without the need to track any MPTCP-related state. To achieve this, the frontend proxy needs to generate K and Z in a specific way. Basically, the proxy derives the token through the method described at the beginning of this Section 3. This gives us the following relation:

token = block_cipher(proxy_id, Y) (Y is the local secret)

However, as described above, at the same time we enforce:

token = block_cipher(Z, K)

Thus, the proxy simply generates a random number K, and can thus generate Z by decrypting the token with key K. It is TBD what number of bits of a token could be used for conveying routing information. Exlcuding those bits, the token would be random, and the key K is random as well, so Z will be random as well. An attacker evesdropping the token cannot infer anything on Z nor on K. However, prolonged gathering of token data could lead to building up some data about the key K.

4. Conclusion

In order to be deployable at a large scale, Multipath TCP has to evolve to accomodate the use-case of distributed layer-4 loadbalancers. In this document we explained the different problems that arise when one wants to deploy MPTCP in a large server farm. We followed up with two possible approaches to solve the issues around the non-deterministic nature of the token. We argue that it is important that the working group considers this problem and strives to find a solution.

5. IANA Considerations

No IANA considerations.

6. References

6.1. Normative References

[I-D.paasch-mptcp-syncookies]
Paasch, C., Biswas, A., and D. Haas, "Making Multipath TCP robust for stateless webservers", draft-paasch-mptcp-syncookies-00 (work in progress), April 2015.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

6.2. Informative References

[Greenberg13]
Greenberg, A., Lahiri, P., Maltz, D., Parveen, P., and S. Sengupta, "Towards a Next Generation Data Center Architecture: Scalability and Commoditization", 2018, <http://dl.acm.org/citation.cfm?id=1397732>.

[Patel13] Parveen, P., Bansal, D., Yuan, L., Murthy, A., Maltz, D., Kern, R., Kumar, H., Zikos, M., Wu, H., Kim, C., and N. Karri, "Ananta: Cloud Scale Load Balancing", 2013, <http://dl.acm.org/citation.cfm?id=2486026>.

[Shuff13] Shuff, P., "Building A Billion User Load Balancer", 2013, <https://www.youtube.com/watch?v=MKgJeqF1DHw>.

Authors' Addresses

Christoph Paasch
Apple, Inc.
Cupertino
US

Email: cpaasch@apple.com

Greg Greenway
Apple, Inc.
Cupertino
US

Email: ggreenway@apple.com

Alan Ford
Pexip

Email: alan.ford@gmail.com