TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Updates: 793 (if approved)
Intended status: Experimental
Expires: March 26, 2015

B. Briscoe
BT

September 22, 2014

Extended TCP Option Space in the Payload of an Alternative SYN
draft-briscoe-tcpm-syn-op-sis-02

Abstract

This document describes an experimental method to extend the option space for connection parameters within the initial TCP SYN segment at the start of a TCP connection. In this method the TCP client sends two alternative SYNs: one intended for legacy servers and one intended for upgraded servers. Once it establishes which type of server has responded, it continues the connection appropriate to that server type and aborts the other. The SYN intended for upgraded servers includes additional options at the end of the payload. It is designed to traverse all known middleboxes. In the longer term, clients will be able to send only the SYN intended for upgraded servers.

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 March 26, 2015.

Table of Contents

1. Introduction

This document describes an experimental method to extend the TCP option space available in the initial SYN segment of a TCP connection (i.e. SYN set and ACK not set) [RFC0793]. This extension is required to support some combinations of TCP options, notably large ones such as TCP AO [RFC5925] (16B), Multipath TCP [RFC6824] (12B), and TCP Fast Open [I-D.ietf-tcpm-fastopen] (6-18B) as well as other options already typically used in TCP connections, such as SACK-ok (2B), Timestamp (10B), Window Scale (3B), MSS (4B) .

In this method the TCP client sends two alternative SYNs: one intended for legacy servers and one intended for upgraded servers. Once it establishes which type of server has responded, it continues the connection appropriate to that server type and aborts the other. The SYN intended for upgraded servers includes additional options at the end of the payload. It is designed to traverse all known middleboxes.

The ambition of this specification is more than just a low latency way to extend the TCP option space using two SYNs for parallel capability negotiation. A larger goal is to enable evolution towards:

- a single TCP initial segment with more space for control options and

- a more structured way for TCP to determine which control options might interact with middleboxes and which are intended solely for end-system interaction.

1.1. Motivation for Adoption (to be removed before publication)

It is recognised that there could be potential for compressing together multiple options in order to mitigate the option space problem. However, it seems inevitable that ultimately more option space will be needed, particularly given that many of the TCP options introduced recently consume large numbers of bits in order to provide sufficient information entropy, which is not amenable to compression.

Extension of TCP option space on a SYN requires support from both ends. This means it will take many years before the facility is functional for most pairs of end-points. Therefore, given the problem is already becoming pressing, a solution needs to start being deployed now.

1.2. Scope

This experimental specification extends the TCP wire protocol. It is independent of the dynamic behaviour of TCP and it is independent of (and thus compatible with) any protocol that encapsulates TCP, including IPv4 and IPv6.

1.3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that provides extra option space on the initial TCP SYN segment. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness and interoperability (with itself, with previous version of TCP, and with various commonly deployed middleboxes).

Success criteria: The experimental protocol will be considered successful if it satisfies the following requirements in the consensus opinion of the IETF tcpm working group. {ToDo: describe success criteria}

Duration: To be credible, the experiment will need to last at least 12 months from publication of the present specification. If successful, a report on the experiment will be written up. it would then be appropriate to work on a standards track specification, in which the experiment report may be included.

1.4. Terminology

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]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

TCP header: As defined in RFC 793 [RFC0793]. Even though the present specification places TCP options at the end of the payload, the term 'TCP header' is still used to mean only those fields at the head of the segment, delimited by the TCP Data Offset.

Extra TCP options: TCP options placed in the space that the present specification makes available beyond the Data Offset, and beyond any optional payload.

TCP payload: User-data to be passed to the layer above TCP. The present document redefines the TCP payload so that it does not include the extra TCP options placed at the end of the payload.

Legacy connection: A connection starting with a SYN that a pre-existing TCP server will fully understand.

Upgraded connection: A connection starting with an upgraded SYN that will only be fully understood by a server complying with the present specification (even though it might appear valid to a pre-existing TCP server).

Legacy server: A TCP listener complying with pre-existing TCP specifications, but not with the present document.

Upgraded server: A TCP listener complying with the present document as well as with pre-existing TCP specifications.

2. Protocol Specification

2.1. Dual 3-Way Handshake

The upgraded TCP client sends two alternative SYNs: a regular SYN in case the server is legacy and a SYN-U (see Section 2.3.1) in case the server is upgraded. The two SYNs MUST have the same network addresses and the same destination port, but different source ports. Once the client establishes which type of server has responded, it continues the connection appropriate to that server type and aborts the other.

The SYN intended for upgraded servers (SYN-U) includes additional TCP options at the end of the payload (see Section 2.3.1). The options are placed at the end of the payload to ensure that the SYN-U is more likely to traverse middleboxes that inspect application-layer headers, which they expect to be at the start of the payload.

Table 1 summarises the TCP 3-way handshake exchange for each of the two SYNs between an upgraded TCP client (the active opener) and either:

1. a legacy server, using the two columns to the left, or

2. an upgraded server, using the two columns to the right

Because the two SYNs come from different source ports, the server will treat them as separate connections, probably using separate threads (assuming a threaded server). A load balancer might forward each SYN to separate replicas of the same logical server. Each

replica will deal with each incoming SYN independently - it does not need to co-ordinate with the other replica.

	Legacy Server Thread X	Legacy Server Thread Y		Upgraded Server Thread X	Upgraded Server Thread Y
1	>SYN	>SYN-U		>SYN	>SYN-U
2	<SYN/ACK	<SYN/ACK		<SYN/ACK	<SYN/ACK-U
3	Client waits for response to both SYNs			Client waits for response to SYN-U	
4	>ACK	>RST		>RST	>ACK
5	Cont...				Cont...

Table 1: Dual 3-Way Handshake in Two Server Scenarios

Each column of the table shows the required 3-way handshake exchange within each connection, using the following symbols:

> means client to server

< means server to client

Cont... means the TCP connection continues as normal

The connection that starts with a regular SYN is called the 'legacy connection' and the one that starts with a SYN-U is called the 'upgraded connection'. An upgraded server MUST respond to a SYN-U with an upgraded SYN/ACK (termed a SYN/ACK-U and defined in Section 2.3.2). Then the client recognises that it is talking to an upgraded server. The client's behaviour depends on which response it receives first, as follows:

- If the client first receives a SYN/ACK response on the legacy connection, it MUST wait for the response on the upgraded connection. It then proceeds as follows:
  - If the response on the upgraded connection is a regular SYN/ACK, the client MUST reset (RST) the upgraded connection and it can continue with the legacy connection.

    * If the response on the upgraded connection is an upgraded SYN/ACK-U, the client MUST reset (RST) the legacy connection and it can continue with the upgraded connection.

- If the client first receives a legacy SYN/ACK response on the upgraded connection, it MUST reset (RST) the upgraded connection immediately. It can then wait for the response on the legacy connection and, once it arrives, continue as normal.

- If the client first receives an upgraded SYN/ACK-U response on the upgraded connection, it MUST reset (RST) the legacy connection immediately and continue with the upgraded connection.

2.2. Retransmission Behaviour

If the client receives a response to the SYN, but a short while after that {duration TBA} the response to the SYN-U has not arrived, it SHOULD retransmit the SYN-U. If latency is more important than the extra TCP options, in parallel to any retransmission, or instead of any retransmission, the client MAY give up on the upgraded (SYN-U) connection by sending a reset (RST) and completing the 3-way handshake of the legacy connection.

If the client receives no response at all to either the SYN or the SYN-U, it SHOULD solely retransmit one or the other, not both. If latency is more important than the extra TCP options, it will retransmit the SYN. Otherwise it will retransmit the SYN-U. It MUST NOT retransmit both segments, because the lack of response could be due to severe congestion.

2.3. Segment Structure

2.3.1. SYN-U Structure (Non-Deterministic)

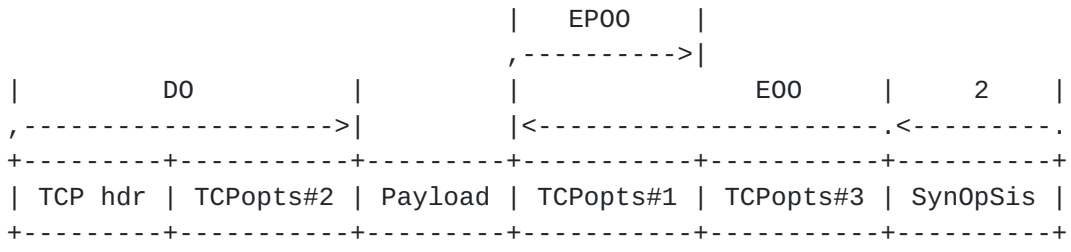
{Temporary note: The structure for a SYN-U segment specified in this section leads to slightly non-deterministic behaviour, so it will be labelled SYN-UN (for Upgraded Non-deterministic). A deterministic alternative is given in Appendix A. It is expected that one will be chosen during the IETF review process, at which point the other will be deleted.}

A SYN-UN is structured as shown in Figure 1. Up to the payload, it is identical to a regular TCP SYN segment, with a base TCP header (TCP hdr) and the usual facility to set the Data Offset (DO) to allow space for TCP options (TCPopts#2). The significance of '#2' will be explained later.

Unlike a legacy TCP segment, the payload of a SYN-UN does not continue to the end of the packet. Instead, it can be seen that space is provided for additional TCP options at the end of the packet at an offset from the end of the packet defined using the Extra Options Offset (EOO) field. The EOO field is read from a new 'SynOpSis' TCP option defined in this specification.

Note that the handshake described earlier (Section 2.1) ensures that a legacy server will never erroneously pass this mixture of payload and options to the application. If a SYN carries a payload, a TCP server holds back the payload from the application until the 3-way handshake completes. And, once the upgraded client recognises it is talking to a legacy server it will abort the 3-way handshake of the upgraded connection. Therefore it will always prevent the mixed payload from confusing the application.

The SynOpSis TCP option MUST be the final TCP option right-aligned at the end of the payload so that the server can find it (using the length of the whole packet found in the network layer header, e.g. IPv4 or IPv6).



All offsets are specified in 4-octet (32-bit) words.

Figure 1: The Structure of a SYN-UN segment (not to scale)

The SynOpSis TCP option has Kind SynOpSis, with a value {TBA} (See Section 7). The internal structure of the SynOpSis TCP option for a SYN-UN is defined in Figure 2. In general, the SynOpSis TCP option can have different lengths for different purposes. However, in a SYN-UN, the SynOpSis TCP option MUST have Length = 8, so that the server can find where it starts (8 octets before the end of the segment). The first 4 octets of the option contain a magic number {TBA} to reduce the chance that arbitrary data within the payload will be mistaken for a SynOpSis TCP option.

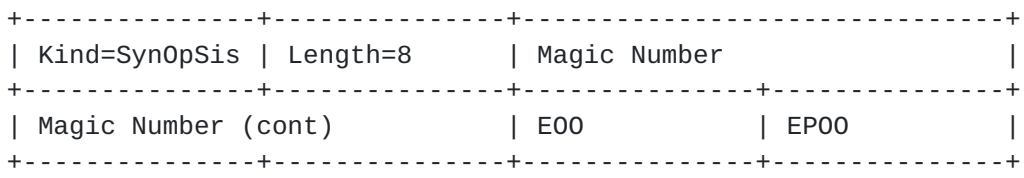


Figure 2: SynOpSis TCP Option for a SYN-UN

Two 1-octet offset fields are placed at the end of the SynOpSis TCP option for a SYN-UN:

The Extra Options Offset (EOO): The EOO field defines the total size of the extra TCP options in 4-octet words. The start of the extra options will be located 4 * (EOO + 2) octets from the end of the packet. The IP payload size will be 4 * (DO + EOO + 2) + TCP_payload_size.

The Extra Prefix Options Offset: The EPOO field defines an additional offset from the start of the extra TCP options that identifies the extent of those extra TCP options that need to be processed before any regular TCP options. The EPOO field defines this offset in 4-octet words.

2.3.2. SYN/ACK-U Structure

The SYN/ACK-U carries a simple SynOpSis flag TCP option as defined in Figure 3. It solely identifies that the SYN/ACK is from a server that supports SynOpSis TCP options.

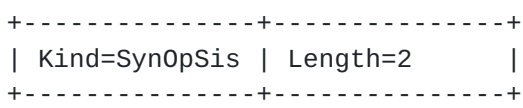


Figure 3: A SynOpSis flag TCP option

2.4. TCP Option Processing

2.4.1. Writing TCP Options

If an upgraded TCP client includes the TCP Fast Open option [I-D.ietf-tcpm-fastopen] in the SYN, it MUST be placed with the extra TCP options after the end of the payload. An upgraded TCP client MUST NOT place any TCP option in the TCP header of a SYN that might cause a TCP server to pass user-data directly to the application before the 3-way handshake completes.

In order to ensure that the first extra TCP option aligns on a 4-octet word boundary, a TCP client SHOULD {ToDo: MUST?} start the extra TCP options with sufficient 1-octet no-op TCP options [RFC0793]. The number of no-op octets required will be 3 - ((S - 1) % 4), where S is the IP payload size in octets and '%' is the modulo operation.

2.4.2. Reading TCP Options

Before processing any TCP options, if the TCP payload is greater than 9 octets, an upgraded server MUST determine whether there is a SynOpSis TCP option at the end of the packet by checking all the following conditions:

- The Kind value is the SynOpSis Kind value;
- The length is 8;
- The next 4 octets match the magic number;
- The sum of the value of the EOO field, and all the length fields found by walking along the TCP options at the end of the payload exactly reaches the end of the packet.

If any of these conditions fails, the server MUST proceed by processing any TCP options in the TCP header (TCPopts#2 in Figure 1), and treat all octets after the Data Offset as user-data.

If an upgraded server finds a valid SynOpSis TCP option at the end of the packet, it MUST process the TCP options in a SYN-UN in the following order:

1. The Prefix TCP options (TCPopts#1 in Figure 1)
2. The regular TCP options following the main header but before the payload (TCPopts#2 in Figure 1);
3. The Suffix TCP options (TCPopts#3 in Figure 1)

This arrangement allows the client to reveal certain TCP options for processing by middleboxes (TCPopts#2), while concealing others after the payload. And the client can still control the order in which the server processes all the TCP options.

2.4.3. Forwarding TCP Options

Middleboxes exist that process some aspects of the TCP header. Although the present specification defines a new location for extra TCP options at the end of a packet, this is intended for the exclusive use of the destination TCP implementation. Legacy middleboxes will not expect to find TCP options beyond the Data Offset anyway. A middlebox MUST continue to treat any data beyond the Data Offset solely as user-data.

A TCP implementation is not necessarily aware whether it is deployed in a middlebox or in a destination, e.g. a split TCP connection might use a regular TCP implementation. Therefore, a general-purpose TCP that implements the present specification will need a configuration switch to disable any search for TCP options at the end of the packet.

3. Discussion of Non-Determinism

All the TCP headers and options before the payload of a SYN-UN (see Section 2.3.1) are completely indistinguishable from a regular SYN. This makes it very likely that a SYN-UN will be able to traverse any legacy middlebox, even one that splits a TCP connection. A SYN-UN can only be distinguished from any legacy SYN by the presence of the SynOpSis bit-pattern at the end of the packet.

This is termed the non-deterministic segment structure, because there will be a very small probability (roughly 2^{-48-L}) that payload data on a regular (non-SynOpSis) SYN could:

- happen to contain a pattern in exactly the right place that matches the kind, length and magic number of a SynOpSis TCP option and

- happen to contain a valid sequence of numbers in exactly the right places to look like a valid sequence of TCP option lengths.

In the above formula, L is the sum of all the bits in all the TCP option length fields that seem to be in the payload. For instance, if it appears that there are 2 TCP options before the SynOpSis option at the end of the payload, then L=2*8=16, and the probability of incorrectly using user-data as TCP options will then be roughly 2^(-64) = 1 in 18 billion billion (18x10^18). This 'stealth' approach has been taken in order to maximise the chances of traversing all the various types of middlebox.

Note that the non-determinism is only in one direction. I.e., there is a small chance that arbitrary user data might be mistaken for the

SynOpSis TCP option, but it is not possible that a valid SynOpSis TCP option would ever be mistaken for user data.

{ToDo: It is recognised that it is potentially unsafe to use probability to determine whether TCP options are hidden at the end of the payload. If the WG prefers not to use the non-deterministic structure in Section 2.3.1, it can be replaced with the alternative more conventional deterministic protocol structure in . (Appendix A.1), and this discussion of non-determinism could then be deleted.}

4. Migration to Single Handshake

The strategy of sending two SYNs in parallel is not essential to the Alternative SYN approach. It is merely an initial strategy that minimises latency when the client does not know whether the server has been upgraded. Evolution to a single SYN with greater optio space could proceed as follows:

- Clients could maintain a white-list of upgraded servers discovered by experience and send just the upgraded SYN-U in these cases.

- Then, for white-listed servers, the client could send a legacy SYN only in the rare cases when an attempt to use an upgraded connection had previously failed (perhaps a mobile client encountering a new blockage on a new path to a server that it had previously accessed over a good path).

- In the longer term, once it can be assumed that most servers are upgraded and the risk of having to fall back to legacy has dropped to near-zero, clients could send just the upgraded SYN first, without maintaining a white-list, but still be prepared to send a legacy SYN in the rare cases when that might fail.

There is concern that, although dual handshake approaches might well eventually migrate to a single handshake, they do not scale when there are numerous choices to be made simultaneously. For instance, trying IPv4 and IPv6 in parallel [RFC6555]; and trying SCTP and TCP in parallel [I-D.wing-tsvwg-happy-eyeballs-sctp]; and trying ECN and non-ECN in parallel; and so on. Nonetheless, it is not necessary to try every possible combination of N choices, which would otherwise require 2^N handshakes (assuming each choice is between two options). Instead, a selection of the choices could be attempted together. At the extreme, two handshakes could be attempted, one with all the new features, and one without all the new features.

5. Interaction with Pre-Existing TCP

{ToDo: TCP API, TCP States and Transitions, TCP Segment Processing, Processing and Segment Size Overhead, Connectionless Resets, ICMP Handling. Interaction with EDO, Interaction with TFO (see Section 2.4.1), Interactions with Other TCP Variants including SYN Cookies, Forward-Compatibility, Interaction with TCP assumptions of Middleboxes. }

6. Dual Handshake: The Explicit Variant

This explicit dual handshake is similar to that in Section 2.1, except the SYN that the client intends for a legacy server is explicitly distinguishable from the SYN that would be sent by a legacy client. Then, in the case of an upgraded server, the server can reset the legacy connection itself, rather than creating connection state for at least a round trip until the client resets the connection.

{Temporary note: The choice between the explicit handshake in the present section or the handshake in Section 2.1 is a tradeoff between robustness against middlebox interference and minimal server state. During the IETF review process, one might be chosen as the only variant to go forward, at which point the other will be deleted. Alternatively, the IETF could allow both variants and a client could be implemented with either, or both. If both, the application could choose which to use at run-time. Then we will need a section describing the necessary API.}

For an explicit dual handshake, the TCP client still sends two alternative SYNs: a SYN-L intended for legacy servers and a SYN-U intended for upgraded servers. The two SYNs MUST have the same network addresses and the same destination port, but different source ports. Once the client establishes which type of server has responded, it continues the connection appropriate to that server type and aborts the other. The SYN intended for upgraded servers includes additional options at the end of the payload (the SYN-U defined as before in Section 2.3.1).

Table 2 summarises the TCP 3-way handshake exchange for each of the two SYNs between an upgraded TCP client (the active opener) and either:

1. a legacy server, using the two columns to the left, or

2. an upgraded server, using the two columns to the right

The table uses the same layout and symbols as Table 1, which have already been explained in Section 2.1.

	Legacy Server Thread X	Legacy Server Thread Y		Upgraded Server Thread X	Upgraded Server Thread Y
1	>SYN-L	>SYN-U		>SYN-L	>SYN-U
2	<SYN/ACK	<SYN/ACK		<RST	<SYN/ACK-U
3	>ACK	>RST			>ACK
4	Cont...				Cont...

Table 2: Explicit Variant of Dual 3-Way Handshake in Two Server Scenarios

As before, an upgraded server MUST respond to a SYN-U with a SYN/ACK-U. Then, the client recognises that it is talking to an upgraded server.

Unlike before, an upgraded server MUST respond to a SYN-L with a RST. However, the client cannot rely on this behaviour, because a middlebox might strip the SynOpSis TCP option from the SYN-L before it reaches the server. Then the handshake would effectively revert to the implicit variant. Therefore the client's behaviour still depends on which SYN-ACK arrives first, so its response to SYN-ACKs has to follow the rules specified for the implicit handshake variant in Section 2.1.

The rules for processing TCP options are unchanged from those in Section 2.4.

6.1. Retransmission Behaviour - Explicit Variant

If the client receives a RST on one connection, but a short while after that {duration TBA} the response to the SYN-U has not arrived, it SHOULD retransmit the SYN-U. If latency is more important than the extra TCP options, in parallel to any retransmission, or instead of any retransmission, the client MAY send a SYN without any SynOpSis option, in case this is the cause of the black-hole. However, the presence of the RST implies that one of the SYNs with a SynOpSis TCP option (the SYN-L) probably reached the server, therefore it is more likely (but not certain) that the lack of response on the other connection is due to transmission loss or congestion loss.

If the client receives no response at all to either the SYN-L or the SYN-U, it SHOULD solely retransmit one or the other, not both. If latency is more important than the extra TCP options, it SHOULD send a SYN without a SynOpSis TCP option. Otherwise it SHOULD retransmit the SYN-U. It MUST NOT retransmit both segments, because the lack of response could be due to severe congestion.

6.2. SYN-L Structure

The SYN-L is merely a SYN with with an extra SynOpSis flag option as shown in Figure 3 (see Section 2.3.2). It solely identifies that the SYN is from a client that supports SynOpSis TCP options. In the case of a legacy server, it will just ignore this TCP option that it doesn't recognise.

6.3. Corner Cases

There is a small but finite possibility that one load-sharing replica of a server is upgraded, while another is not. The Implicit Handshake is robust to this possibility, but the Explicit Handshake is not., unless the following additional rules are followed:

Both aborted: The client might receive a RST on its legacy connection in response to its SYN-L, then a regular SYN/ACK on its upgraded connection in response to its SYN-U. In this case, the client MUST still respond with a RST on its upgraded connection. Otherwise, its extra TCP options will be passed as user-data to the application by the legacy server. If confronted with this unusual scenario where both connections are aborted, the client's only recourse is to retry a new dual handshake on different source ports, or ultimately to fall-back to sending a regular SYN.

Both successful: This could happen in either order but, in both cases, the client aborts the last connection to respond:

* The client completes the legacy handshake (because it receives a SYN/ACK), but then, before it has aborted the upgraded connection, it receives a SYN/ACK-U on it. In this case, the client MUST abort the upgraded connection even though it would work. Otherwise the client will have opened both connections, one with extra TCP options and one without. This could confuse the application.

* The client completes the the upgraded connection after receiving a SYN/ACK-U, but then it receives a SYN/ACK on the legacy connection. In this case, the client MUST abort the legacy connection.

7. IANA Considerations

This specification requires IANA to allocate one value from the TCP option Kind name-space, against the name "Sister SYN Options (SynOpSis)"

Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and magic number 0xHHHH (16 bits) {ToDo: Value TBA and register this with IANA}, then migrate to the new option after the allocation.

8. Security Considerations

Certain cryptographic functions have different coverage rules for the TCP header and TCP payload. Placing some TCP options at the end of the payload could mean that they are treated differently from regular TCP options. This is a deliberate feature of the protocol, but application developers will need to be aware that this is the case.

{ToDo: More}

9. Acknowledgements

The idea of this approach grew out of discussions with Joe Touch while developing draft-touch-tcpm-syn-ext-opt, and with Jana Iyengar and Olivier Bonaventure. The idea that it is architecturally preferable to place a protocol extension behind a higher layer, and code its location into upgraded implementations, was originally articulated by Rob Hancock. The following people provided useful review comments: Joe Touch, Yuchung Cheng.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Trilogy 2 project (ICT-317756). The views expressed here are solely those of the authors.

10. References

10.1. Normative References

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

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

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013.

10.2. Informative References

[I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09 (work in progress), July 2014.

[I-D.wing-tsvwg-happy-eyeballs-sctp]
Wing, D. and P. Natarajan, "Happy Eyeballs: Trending Towards Success with SCTP", draft-wing-tsvwg-happy-eyeballs-sctp-02 (work in progress), October 2010.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.

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

Appendix A. Alternative Protocol Specifications

This appendix is informative and will be deleted before publication. It documents protocol alternatives that the IETF may wish to consider in place of those in the body of the specification.

A.1. SYN-U Structure (Deterministic)

This appendix describes a structure for an upgraded SYN called SYN-UD (for upgraded deterministic) that is an alternative to the non-deterministic structure defined in Section 2.3.1. It is termed 'deterministic' because it uses the conventional placement for the SynOpSis TCP option (instead of the unconventional SYN-UN placement at the end of the packet, where arbitrary user-data could be mistaken for the SynOpSis option).

However, given it uses the new SynOpSis TCP option in the TCP header, it will not always successfully traverse middleboxes. Unlike a SYN-UN, a SYN-UD will certainly not traverse legacy middleboxes that do not forward unrecognised TCP options, and it is unlikely to traverse a legacy middlebox that splits TCP connections, unless it copies unrecognised TCP options. Nonetheless, like the SYN-UN, the options are still placed at the end of the payload to ensure that the SYN-UD is more likely to traverse middleboxes that inspect application-layer headers, which they expect to be at the start of the payload.

The placement of the SynOpSis TCP option in a SYN-UD segment is shown in Figure 4. It can be seen that extra TCP options are still placed at the end of the payload at an offset from the end of the packet defined using the Extra Options Offset (EOO) field.

The EOO field is read from a new 'SynOpSis' TCP option defined in this specification. The SynOpSis TCP options is placed in the regular TCP option space of the SYN-UD.

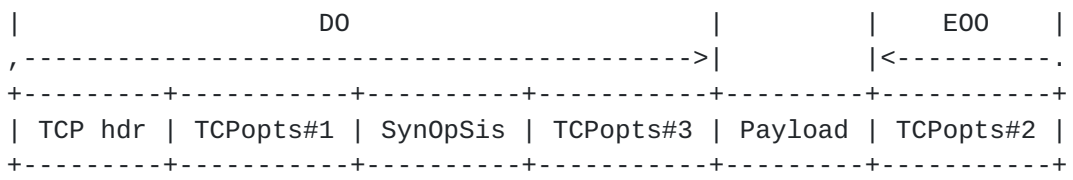


Figure 4: The Structure of an alternative (deterministic) SYN-UD segment (not to scale)

The SynOpSis TCP option for a SYN-UD segment MUST have Kind SynOpSis, with a value {TBA} (See Section 7) and Length = 3. In general, the SynOpSis TCP option can have different lengths for different purposes. However, in a SYN-UD, the SynOpSis TCP option has Length = 3, so that it can carry the 1-octet EOO field, which MUST be present in a SYN-UD. The internal structure of the SynOpSis TCP option for a SYN-UD segment is defined in Figure 5.

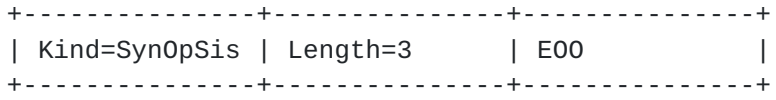


Figure 5: SynOpSis TCP Option for a deterministic SYN-UD

The Extra Options Offset (EOO) field defines the total size of the extra TCP options in 4-octet words. The start of the extra options will be located 4 * EOO octets from the end of the packet. The IP packet payload size will be 4 * (DO + EOO) + TCP_payload_size.

An upgraded server MUST process the TCP options in a SYN-UD in the following order:

1. The regular TCP options following the main header but before the SynOpSis TCP option (TCPopts#1 in Figure 4)

2. The TCP options at the end of the payload (TCPopts#2 in Figure 4)

3. The regular TCP options following the main header but after the SynOpSis TCP option (TCPopts#3 in Figure 4);

Appendix B. Comparison of Alternatives

B.1. Implicit vs Explicit Dual Handshake

In the body of this specification, two variants of the dual handshake are defined:

1. The implicit dual handshake (Section 2.1) with just a regular SYN (no SynOpSis flag option) on the legacy connection;

2. The explicit dual handshake (Section 6) with a SYN-L (SynOpSis flag option) on the legacy connection.

Both schemes double up connection state (for a round trip) on the legacy server. But only the implicit scheme doubles up connection state (for a round trip) on the upgraded server as well. On the other hand, the explicit scheme risks delay accessing a legacy server if a middlebox discards the SYN-L (e.g. some firewalls discard packets with unrecognised TCP options). Table 3 summarises these points.

	SYN (Implicit)	SYN-L (Explicit)
Minimum state on upgraded server	-	+
Minimum risk of delay to legacy server	+	-

Table 3: Comparison of Implicit vs. Explicit Dual Handshake on the Legacy Connection

There is no need for the IETF to choose between these. If the spec allows either or both, the tradeoff can be left to implementers at build-time, or to the application at run-time.

Initially clients might choose the Implicit Dual Handshake to minimise delays due to middlebox interference. But later, perhaps once more middleboxes support the scheme, clients might choose the Explicit scheme, to minimise state on upgraded servers.

B.2. Non-Deterministic vs Deterministic SYN-U

Two alternative segment structures for the SYN-U are defined, but in this case it is recommended that the IETF needs to choose between them so that only one or the other would be specified:

a. The non-deterministic SYN-UN (Section 2.3.1), with the SynOpSis TCP option located at the end of the packet;

b. The deterministic SYN-UD (Appendix A.1), with the SynOpSis TCP option located conventionally in the sequence of TCP options in the TCP header.

The non-deterministic SYN-UN presents a small risk of user data being mistaken for TCP options. Also, whether or not the client needs extra option space, it requires the server to always check for a TCP option at the end of any SYN with a payload greater than 9 octets. On the other hand, the deterministic SYN-UD risks delay accessing an upgraded server because it is visible to middleboxes that discard packets with unrecognised TCP options. Also the SYN-UD is vulnerable to being removed by middleboxes that do not forward unrecognised options, whereas the SYN-UN is likely to traverse all legacy middleboxes, even split TCP connections. Table 4 summarises these points.

	SYN-UN (Non-deterministic)	SYN-UD (Deterministic)
User data unmistakable	-	+
No need for upgraded server to check end of every SYN payload	-	+
Minimum risk of delay to upgraded server	+	-
Extra TCP options likely to traverse all middleboxes	+	-

Table 4: Comparison of Implicit vs. Explicit Dual Handshake on the Legacy Connection

The IETF needs to choose between SYN-UN and SYN-UD, because if implementation of either or both were allowed, the two deficiencies

of SYN-UN would still affect server implementations, whether or not the client used a SYN-UN to take advantage of the two benefits.

Currently this document favours SYN-UN, because SYN-UD's lack of reliable middlebox traversal introduces a functional deficiency (if extra option space is absolutely required, the connection cannot even start). In contrast, SYN-UN's first failing has vanishingly small probability, and its second failing 'only' increases server processing - it does not impair the ability of connections to function outright.

B.3. Comparison with Other Proposals

{ToDo}

Appendix C. Protocol Design Issues (to be Deleted before Publication)

This appendix is informative, not normative. It records outstanding issues with the protocol design that will need to be resolved before publication.

Reliance on segmentation boundary: The definition of the position of the SynOpSis TCP options depends on where the sender decided to place a segment boundary. In general, a sender cannot rely on segment boundaries being preserved, e.g. by segmentation offloading hardware. In the case of a SYN, no more payload data is sent in the first round trip, therefore using this segment boundary is probably safe. However, it may constrain future attempts to send additional data in the first round.

Tie to EDO?: Consider whether a successful SYN/ACK-U implies EDO is also supported.

Size of SynOpSis magic number: Justify choice.

Appendix D. Change Log (to be Deleted before Publication)

A detailed version history can be accesssed at <http://datatracker.ietf.org/doc/draft-briscoe-tcpm-syn-op-sis/history/>

From briscoe...-01 to briscoe...-02:

Technical changes:

* Defined the client behaviour dependent on which response arrives first.

* Allowed retransmission of either SYN or SYN-U if no response from either.

* Redefined EOO as an offset from the end of the packet, not from the beginning of the payload.

* Added section on Migration to a Single Handshake. Reworded dual handshake so that it is not mandatory for the client to send dual SYNs simultaneously; only the relation between the SYNs and the response to either is mandatory, while parallel SYNs is purely for latency reduction.

* Added rules for writing TCP options, i.e. i) options like TFO MUST NOT be located in the TCP header and ii) add no-ops to align on 4-octet boundary.

* Added rules for forwarding TCP options, i.e. only the destination looks for TCP options after the Data Offset, not middleboxes.

* Moved the Explicit Handshake variant (SYN-L) into the body from the appendix, and recommended the choice could be down to implementers or apps. Included section on corner cases.

* Introduced more normative language throughout the Protocol Spec.

Editorial changes:

* Added temporary motivation section

* Added confusible terminology to Terminology section.

* Divided protocol spec into sub-sections.

* Handshake table: Clarified that the two columns under each server represent separate threads, that may run on separate servers, without co-ordination. Represented message dependencies in the alignment of the rows.

* Explained the table.

* Explained why a legacy server won't ever pass SYN-U to the app.

* More precisely described loss as 'not arrived before a timeout', and explained the tradeoff between latency and extra TCP options.

* Gave reasoning for locating TCP options in three groups.

* Acknowledged Rob Hancock for the architectural idea of hiding an extension to a protocol in the layer above.

* Appendix about protocol alternatives now only presents the SYN-UD alternative, given the implicit/explicit handshake choice has been moved to the body.

* Rewrote appendix about comparing the choices to treat the two pairs of choices separately, rather than discussing all four combinations of pairs of choices.

From briscoe...-00 to briscoe...-01:

Technical changes:

* Added the definition of a SYN/ACK-U

* Deterministic Protocol Spec: Replaced SYN/ACK-L with RST (Joe Touch)

* Added Non-Deterministic Explicit and Deterministic Implicit Protocol Specs in Appendices

* Added Comparison of Alternatives as an Appendix

* Security Considerations: Added note about crypto coverage of TCP options in the payload being different from that of other TCP options.

* Added an appendix to record outstanding Protocol Design Issues, and included segmentation boundary issue (Yuchung Cheng).

Editorial changes:

* Changed TCP option Kind from SYN-OP-SIS to SynOpSis

* Protocol Spec: Explained why the extra TCP options are placed at the end of the payload

* Throughout: avoided the ambiguity in the word payload, now that there are TCP options at the end of the payload. Some might consider these to be within the payload, while others might consider them to be placed beyond the payload.

* Segment structure figures: Clarified that they are not to scale.

* Added placeholder section "Interaction with TCP"

* Acknowledged reviewers

Author's Address

Bob Briscoe
BT
B54/77, Adastral Park
Martlesham Heath
Ipswich IP5 3RE
UK

Phone: +44 1473 645196
Email: bob.briscoe@bt.com
URI: http://bobbriscoe.net/