TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Updates: 793 (if approved)
Intended status: Experimental
Expires: April 30, 2015

B. Briscoe
BT

October 27, 2014

Inner Space for TCP Options
draft-briscoe-tcpm-inner-space-01

Abstract

This document describes an experimental method to extend the limited space for control options in every segment of a TCP connection. It can use a dual handshake so that, from the very first SYN segment, extra option space can immediately start to be used optimistically. At the same time a dual handshake prevents a legacy server from getting confused and sending the control options to the application as user-data. The dual handshake is only one strategy - a single handshake will usually suffice once deployment has got started. The protocol is designed to traverse most known middleboxes including connection splitters, because it sits wholly within the TCP Data. It also provides reliable ordered delivery for control options. Therefore, it should allow new TCP options to be introduced i) with minimal middlebox traversal problems; ii) with incremental deployment from legacy servers; iii) without an extra round of handshaking delay iv) without having to provide its own loss recovery and ordering mechanism and v) without arbitrary limits on available space.

Table of Contents

1. Introduction 3
1.1. Motivation for Adoption Now (to be removed before publication) 6
1.2. Scope 6
1.3. Experiment Goals 6
1.4. Document Roadmap 7
1.5. Terminology 7
2. Protocol Specification 9
2.1. Protocol Interaction Model 9
2.1.1. Dual 3-Way Handshake 9
2.1.2. Dual Handshake Retransmission Behaviour 11
2.1.3. Continuing the Upgraded Connection 12
2.2. Upgraded Segment Structure and Format 12
2.2.1. Structure of an Upgraded Segment 12
2.2.2. Format of the InSpace Option 14
2.3. Inner TCP Option Processing 15
2.3.1. Writing Inner TCP Options 15
2.3.1.1. Constraints on TCP Fast Open 15
2.3.1.2. Option Alignment 16
2.3.1.3. Sequence Space Coverage 16
2.3.1.4. Presence or Absence of Payload 16
2.3.2. Reading Inner TCP Options 16
2.3.2.1. Reading Inner TCP Options (SYN=1) 17
2.3.2.2. Reading Inner TCP Options (SYN=0) 18
2.3.3. Forwarding Inner TCP Options 19
2.4. Exceptions 20
2.5. SYN Flood Protection 20
3. Design Rationale 21
3.1. Dual Handshake and Migration to Single Handshake 21
3.2. In-Band Inner Option Space 22
3.2.1. Non-Deterministic Magic Number Approach 22

3.2.2. Non-Goal: Security Middlebox Evasion 23
3.2.3. Avoiding the Start of the First Two Segments 24
3.2.4. Control Options Within Data Sequence Space 24
3.2.5. Rationale for the Sent Payload Size Field 26
3.3. Rationale for the InSpace Option Format 26
3.4. Protocol Overhead 27
4. Interaction with Pre-Existing TCP Implementations 29
4.1. Compatibility with Pre-Existing TCP Variants 29
4.2. Interaction with Middleboxes 31
4.3. Interaction with the Pre-Existing TCP API 31
5. IANA Considerations 33
6. Security Considerations 34
7. Acknowledgements 34
8. References 35
8.1. Normative References 35
8.2. Informative References 35
Appendix A. Protocol Extension Specifications 36
A.1. Disabling InSpace and Generic Connection Mode Switching . 37
A.2. Dual Handshake: The Explicit Variant 39
A.2.1. SYN-O Structure 41
A.2.2. Retransmission Behaviour - Explicit Variant 41
A.2.3. Corner Cases 42
A.2.4. Workround if Data in SYN is Blocked 43
A.3. Jumbo InSpace TCP Option (only if SYN=0) 44
A.4. Upgraded Segment Structure to Traverse DPI boxes 44
Appendix B. Comparison of Alternatives 46
B.1. Implicit vs Explicit Dual Handshake 46
Appendix C. Protocol Design Issues (to be Deleted before Publication) 47
Appendix D. Change Log (to be Deleted before Publication) . . . 48
Author's Address 49

1. Introduction

TCP has become hard to extend, partly because the option space was limited to 40B when TCP was first defined [RFC0793] and partly because many middleboxes only forward TCP headers that conform to the stereotype they expect.

This specification ensures new TCP capabilities can traverse most middleboxes by tunnelling TCP options within the TCP Data as 'Inner Options' (Figure 1). Then the TCP receiver can reconstruct the Inner Options sent by the sender, even if the middlebox resegments the data stream and even if it strips 'Outer' options from the TCP header that it does not recognise. The two words 'Inner Space' are appropriate as a name for the scheme; 'Inner' because it encapsulates options within the TCP Data and 'Space' because the space within the TCP Data is virtually unlimited--constrained only by the maximum segment size.

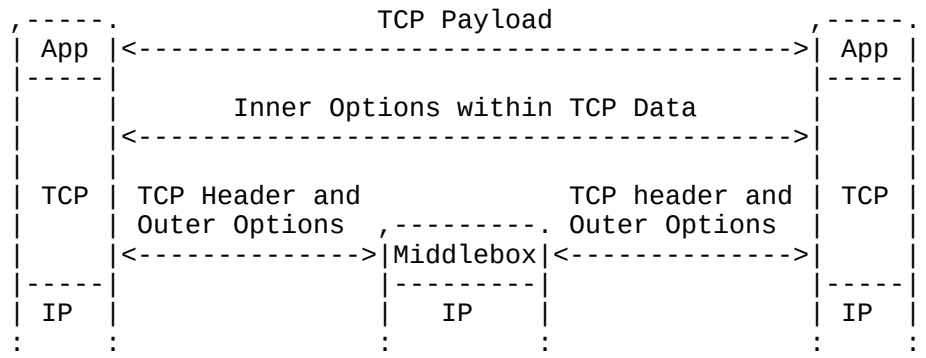


Figure 1: Encapsulation Approach

TCP options fall into three main categories:

a. Those that have to remain as Outer Options--typically those concerned with transmission of each TCP segment, e.g. Timestamps and Selective ACKnowledgements (SACK);

b. Those that are best as Inner Options--typically those concerned with transmission of the data as a stream, e.g. the TCP Authentication Option [RFC5925] or tcpcrypt [I-D.bittau-tcpinc];

c. Those that can be either Inner or Outer Options--typically those used at the start of a connection which is also inherently the start of the first segment so segmentation is not a concern.

Pressure of space is most acute in the initial segments of each half-connection, i.e. the SYN and SYN/ACK, and particularly the SYN. Even though Inner Space is not suitable for category (a) options, moving all of categories (b) and (c) into Inner Space frees up plenty of outer space in the header for category (a).

The following list of options that might be required on a SYN illustrates how acute the problem is:

- 4B: Maximum Segment Size (MSS) [RFC0793];
- 2B: SACK-ok [RFC2018];
- 3B: Window Scale [RFC7323];
- 10B: Timestamp [RFC7323];
- 12B: Multipath TCP [RFC6824];

- 6-18B: TCP Fast Open on a resumed connection [I-D.ietf-tcpm-fastopen];

- 16B: TCP-AO [RFC5925];

There is probably potential for compressing together multiple options in order to mitigate the option space problem. However, the option space problem has to be faced, because complex special placement is already being contemplated for options that can be larger than 40B on their own (e.g. the key agreement options of tcpcrypt [I-D.bittau-tcpinc]).

Given the Inner Space protocol places control options within TCP Data, it is critical that a legacy TCP receiver is never confused into passing this mix to an application as if it were pure data. Naively, both ends could handshake to check they understand the protocol, but this would introduce a round of delay and it would not solve the shortage of space in a SYN. Instead, the client uses dual handshakes; one suitable for an upgraded server, and the other for an ordinary server. Then, if the client discovers that the server does not understand the new protocol, it can abort the upgraded handshake before the server passes corrupt data to the application. Otherwise, if the server does understand the new protocol, the client can abort the ordinary handshake. Either way, it has added zero extra delay. Interworking of the dual handshake with TCP Fast Open [I-D.ietf-tcpm-fastopen] is carefully defined so that either server can pass data to the application as soon as the initial SYN arrives.

When control options are placed within the TCP Data they inherently get delivered reliably and in order. Although this was not originally recognised as part of the design brief, it offers the significant benefit of simplifying the design of new TCP options. Reliable ordered delivery no longer has to be individually crafted into the design of each new TCP option.

Solving the five problems of i) option-space exhaustion; ii) middlebox traversal; iii) legacy server confusion; iv) reliable ordered control message delivery; and v) handshake latency; does not come without cost:

- So that the Inner Space protocol is immune to option stripping, it flags its presence using a magic number within the TCP Data of the initial segment in each direction, not a conventional TCP option in the header. This introduces a risk that payload in an ordinary SYN or SYN/ACK might be mistaken for the Inner Space protocol (an initial worst-case estimate of the probability is one connection globally every 40 years). Nonetheless, the risk is zero in the (currently common) case of an ordinary connection without payload

during the handshake. There is also no risk of a mistake the other way round--an upgraded connection cannot be mistaken for an ordinary connection.

- Although the dual handshake introduces no extra latency, it introduces extra connection processing & state, extra traffic and extra header processing. Initial estimates put the percentage overhead in single digits for connection processing and state, and traffic overhead at only a few hundredths of a percent. Nonetheless, once the most popular TCP servers have upgraded, only a single handshake will be necessary most of the time and overhead should drop to vanishingly small proportions.

Finally, it should be noted that the ambition of this work is more than just an incrementally deployable, low latency way to extend TCP option space. The aim is to move towards a more structured way for middleboxes to interact transparently with, rather than arbitrarily interfere with, end-system TCP stacks. This has been achieved for connection and stream control options, but it will still be hard to introduce new per-segment control options, which will still have to be located within the traditional Outer TCP Options.

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

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 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 rate control 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.

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. The protocol needs to be sufficiently well specified so that more than one implementation can be built in order to test its function, robustness, overhead and interoperability (with itself, with previous version of TCP, and with various commonly deployed middleboxes). Non-functional issues such as recommendations on message timing also need to be tested. Various optional extensions to the protocol are proposed in Appendix A so experiments are also needed to determine whether these extensions ought to remain optional, or perhaps be removed or become mandatory.

Duration: To be credible, the experiment will need to last at least 12 months from publication of the present specification. If successful, it would then be appropriate to progress to a standards track specification, complemented by a report on the experiments.

1.4. Document Roadmap

The body of the document starts with a full specification of the Inner Space extension to TCP (Section 2). It is rather terse, answering 'What?' and 'How?' questions, but deferring 'Why?' to Section 3. The careful design choices made are not necessarily apparent from a superficial read of the specification, so the Design Rationale section is fairly extensive. The body of the document ends with Section 4 that checks possible interactions between the new scheme and pre-existing variants of TCP, including interaction with partial implementations of TCP in known middleboxes.

Appendix A specifies optional extensions to the protocol that will need to be implemented experimentally to determine whether they are useful. And Appendix B discusses the merits of the chosen design against alternative schemes.

1.5. 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 [RFC0793]. Even though the present specification places TCP options beyond the Data Offset, the term 'TCP Header' is still used to mean only those fields at the head of the segment, delimited by the TCP Data Offset.

Inner TCP Options (or just Inner Options): TCP options placed in the space that the present specification makes available beyond the Data Offset.

Outer TCP Options (or just Outer Options): The TCP options in the traditional location directly after the base TCP Header and before the TCP Data Offset.

Prefix TCP Options: Inner Options to be processed before the Outer Options.

Suffix TCP Options: Inner Options to be processed after the Outer Options.

TCP options: Any TCP options, whether inner, outer or both. This specification makes this term on its own ambiguous so it should be qualified if it is intended to mean TCP options in a certain location.

TCP Payload: Data to be passed to the layer above TCP. The present specification redefines the TCP Payload so that it does not include the Inner TCP Options, the Inner Space Option and any Magic Number, even though they are located beyond the Data Offset.

TCP Data: The information in a TCP segment after the Data Offset, including the TCP Payload, Inner TCP Options, the Inner Space Option and the Magic Number defined in the present specification.

client: The process taking the role of actively opening a TCP connection.

server: The process taking the role of TCP listener.

Upgraded Segment: A segment that will only be fully understood by a host complying with the present specification (even though it might appear valid to a pre-existing TCP receiver). Similarly, Upgraded SYN, Upgraded SYN/ACK etc.

Ordinary Segment: A segment complying with pre-existing TCP specifications but not the present specification. Similarly, Ordinary SYN, Ordinary SYN/ACK etc.

Upgraded Connection: A connection starting with an Upgraded SYN.

Ordinary Connection: A connection starting with an Ordinary SYN.

Upgraded Host: A host complying with the present document as well as with pre-existing TCP specifications. Similarly Upgraded TCP Client, Upgraded TCP Server, etc.

Legacy Host: A host complying with pre-existing TCP specifications, but not with the present document. Similarly Legacy TCP Client, Legacy TCP Server, etc.

Note that the term 'Ordinary' is used for segments and connections, but the term 'Legacy' is used for hosts. This is because, if the Inner Space protocol were widely used in future, a host that could not open an Upgraded Connection would be considered deficient and therefore 'Legacy', whereas an Ordinary Connection would not be considered deficient in the future; because it will always be legitimate to open an Ordinary Connection if extra option space is not needed.

2. Protocol Specification

2.1. Protocol Interaction Model

2.1.1. Dual 3-Way Handshake

During initial deployment, an Upgraded TCP Client sends two alternative SYNs: an Ordinary SYN in case the server is legacy and a SYN-U 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 without completing the 3-way handshake.

The format of the SYN-U will be described later (Section 2.2.2). At this stage it is only necessary to know that the client can put either TCP options or payload (or both) in a SYN-U, in the space traditionally intended only for payload. So if the server's response shows that it does not recognise the Upgraded SYN-U, the client is responsible for aborting the Upgraded Connection. This ensures that a Legacy TCP Server will never erroneously confuse the application by passing it TCP options as if they were user-data.

Section 3.1 explains various strategies the client can use to send the SYN-U first and defer or avoid sending the Ordinary SYN. However, such strategies are local optimizations that do not need to be standardized. The rules below cover the most aggressive case, in which the client sends the SYN-U then the Ordinary SYN back-to-back to avoid any extra delay. Nonetheless, the rules are just as applicable if the client defers or avoids sending the Ordinary SYN.

Table 1 summarises the TCP 3-way handshake exchange for each of the two SYNs in the two right-hand columns, between an Upgraded TCP Client (the active opener) and either:

1. a Legacy Server, in the top half of the table (steps 2-4), or

2. an Upgraded Server, in the bottom half of the table (steps 2-4)

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.

		Ordinary Connection	Upgraded Connection
1	Upgraded Client	>SYN	>SYN-U
/\/\	/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\
2	Legacy Server	<SYN/ACK	<SYN/ACK
3a	Upgraded Client	Waits for response to both SYNs	
3b	"	>ACK	>RST
4		Cont...	
/\/\	/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\
2	Upgraded Server	<SYN/ACK	<SYN/ACK-U
3a	Upgraded Client	Waits for response to SYN-U	
3b	"	>RST	>ACK
4			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.

The connection that starts with an Ordinary SYN is called the 'Ordinary 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 2.2.2). 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 Ordinary Connection, it MUST wait for the response on the Upgraded Connection. It then proceeds as follows:
  - If the response on the Upgraded Connection is an Ordinary SYN/ACK, the client MUST reset (RST) the Upgraded Connection and it can continue with the Ordinary Connection.
  - If the response on the Upgraded Connection is an Upgraded SYN/ACK-U, the client MUST reset (RST) the Ordinary Connection and it can continue with the Upgraded Connection.
- If the client first receives an Ordinary SYN/ACK response on the Upgraded Connection, it MUST reset (RST) the Upgraded Connection immediately. It can then wait for the response on the Ordinary 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 Ordinary Connection immediately and continue with the Upgraded Connection.

2.1.2. Dual Handshake Retransmission Behaviour

If the client receives a response to the SYN, but a short while after that {ToDo: 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 option space, 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 Ordinary 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 option space, 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.

2.1.3. Continuing the Upgraded Connection

Once an Upgraded Connection has been successfully negotiated in the SYN, SYN/ACK exchange, either host can allocate any amount of the TCP Data space in any subsequent segment for extra TCP options. In fact, the sender has to use the upgraded segment structure in every subsequent segment of the connection that contains non-zero TCP Payload. The sender can use the upgraded structure in a segment carrying no user-data (e.g. a pure ACK), but it does not have to.

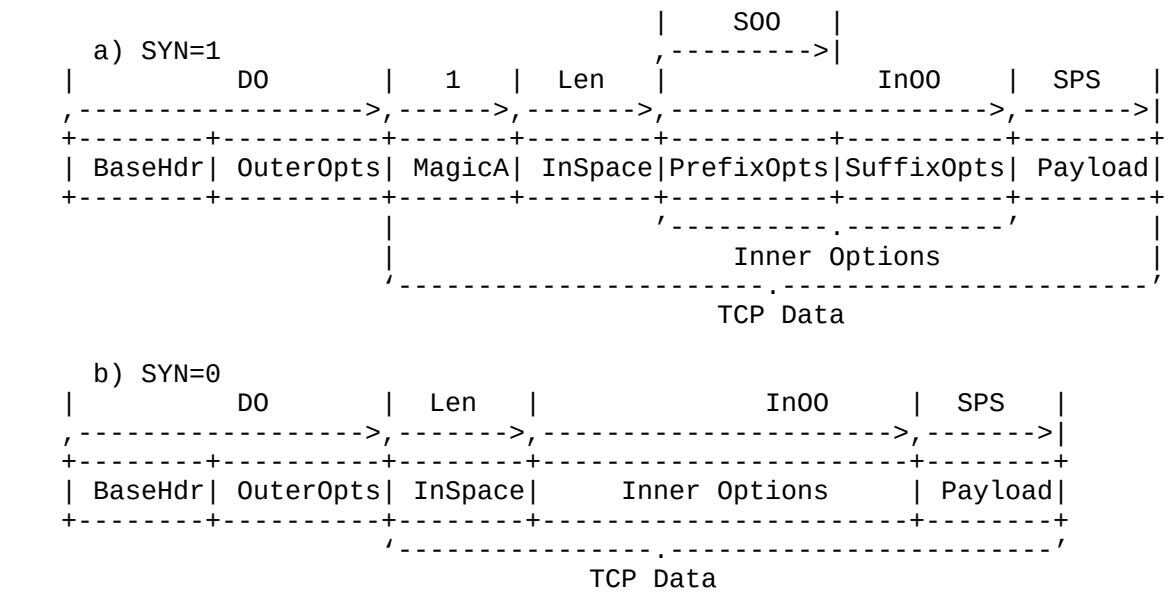
As well as extra option space, the facility offers other advantages, such as reliable ordered delivery of Inner TCP Options on empty segments and more robust middlebox traversal. If none of these features is needed, at any point the facility can be disabled for the rest of the connection, using the ModeSwitch TCP option in Appendix A.1. Interestingly, the ModeSwitch options itself can be very simple because it uses the reliable ordered delivery property of Inner Options, rather than having to cater for the possibility that a message to switch to disabled mode might be lost or reordered.

2.2. Upgraded Segment Structure and Format

2.2.1. Structure of an Upgraded Segment

An Upgraded Segment is structured as shown in Figure 2. Up to the TCP Data Offset, the structure is identical to an Ordinary TCP Segment, with a base TCP Header (BaseHdr) and the usual facility to set the Data Offset (DO) to allow space for TCP options. These regular TCP options are renamed by this specification to Outer TCP Options or just Outer Options, and labelled as OuterOpts in the figure.

The first segment in each direction (i.e. the SYN or the SYN/ACK) is identifiable as upgraded by the presence of the 4-octet Magic Number A (MagicA) at the start of the TCP Data. The probability that an Upgraded Server will mistake arbitrary data at the beginning of the payload of an Ordinary Segment for the Magic Number has to be allowed for, but it is vanishingly small (see Section 3.2.1). Once an Upgraded Connection has been negotiated during the SYN - SYN/ACK exchange, a magic number is not needed to identify Upgraded Segments, because both ends know that the protocol requires the sender to use the upgraded format on all subsequent segments with non-zero TCP Data. Aside from the magic number, the structure of the rest of an Upgraded Segment is effectively the same whether a) SYN=1 or b) SYN=0.



All offsets are specified in 4-octet (32-bit) words, except SPS, which is in octets.

Figure 2: The Structure of an Upgraded Segment (not to scale)

Unlike an Ordinary TCP Segment, the Payload of an Upgraded Segment does not start straight after the TCP Data Offset. Instead, Figure 2 shows that space is provided for additional Inner TCP Options before the TCP Payload. The size of this space is termed the Inner Options Offset (InOO). The TCP receiver reads the InOO field from the Inner Option Space (InSpace) option defined in Section 2.2.2.

The InSpace Option is located in a standardized location so that the receiver can find it:

- On a segment with SYN=1, an Upgraded TCP Sender MUST locate the InSpace Option straight after the magic number, specifically 4 * (DO + 1) octets from the start of the segment.

- On a segment with SYN=0, an Upgraded TCP Sender MUST locate the InSpace Option at the beginning of the TCP Data, specifically 4 * DO octets from the start of the segment.

Because the InSpace Option is only ever located in a standardized location it does not need to follow the RFC 793 format of a TCP option. Therefore, although we call InSpace an 'option', we do not describe it as a 'TCP option'.

The Sent Payload Size (SPS) is also read from within the InSpace Option. If the byte-stream has been resegmented, it allows the receiver to step from one InSpace Option to the next even if the InSpace Options are no longer at the start of each segment (see Section 2.3).

On a segment with SYN=1 (i.e. a SYN or SYN/ACK) the Suffix Options Offset (SOO) is also read from within the InSpace Option. It delineates the end of the Prefix TCP Options (PrefixOpts in the figure) and the start of the Suffix TCP Options (SuffixOpts). When SYN=1, the receiver processes PrefixOpts before OuterOpts, then SuffixOpts afterwards. When SYN=0, the receiver processes the Outer Options before the Inner Options. Full details of option processing are given in Section 2.3.

2.2.2. Format of the InSpace Option

The internal structure of the InSpace Option for an Upgraded SYN or SYN/ACK segment (SYN=1) is defined in Figure 3a) and for a segment with SYN=0 in Figure 3b).

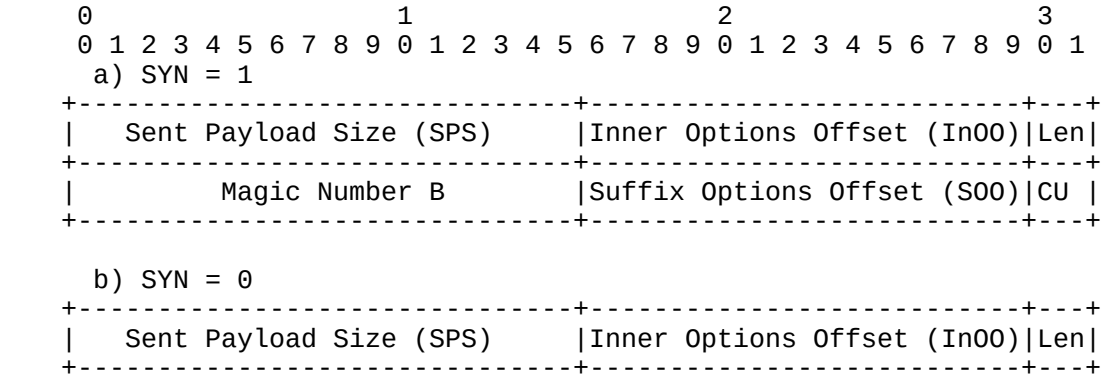


Figure 3: InSpace Option Format

The fields are defined as follows (see Section 3.3 for the rationale behind these format choices):

Option Length (Len): The 2-bit Len field specifies the length of the InSpace Option in 4-octets words (see Section 3.3 for rationale). For this experimental specification:

When SYN=1: the sender MUST use Len=2;

When SYN=0: the sender MUST use Len=1.

Sent Payload Size (SPS): In this 16-bit field the sender MUST record the size in octets of the TCP Payload when it was sent. This specification defines the TCP Payload as solely the user-data to be passed to the application. This excludes Inner TCP options, the InSpace Option and any magic number.

Inner Options Offset (InOO): This 14-bit field defines the total size of the Inner TCP Options in 4-octet words.

The following fields are only defined on a segment with SYN=1 (i.e. a SYN or SYN/ACK):

Magic Number B: The sender MUST fill this 16-bit field with Magic Number B {ToDo: Value TBA} to further reduce the chance that a receiver will mistake the end of an arbitrary Ordinary Payload for the InSpace Option.

Suffix Options Offset (SOO): The 14-bit SOO field defines an additional offset in 4-octet words from the start of the Inner Options that identifies the extent of the Prefix Options (see Section 2.3.2).

Currently Unused (CU): The sender MUST fill the CU field with zeros and they MUST be ignored and forwarded unchanged by other nodes, even if their value is different.

2.3. Inner TCP Option Processing

2.3.1. Writing Inner TCP Options

2.3.1.1. Constraints on TCP Fast Open

If an Upgraded TCP Client uses a TCP Fast Open (TFO) cookie [I-D.ietf-tcpm-fastopen] in an Upgraded SYN-U, it MUST place the TFO option within the Inner TCP Options, beyond the Data Offset.

This rule is specific to TFO, but it can be generalised to any capability similar to TFO as follows: An Upgraded TCP Client MUST NOT place any TCP option in the Outer TCP Options of a SYN if it might cause a TCP server to pass user-data directly to the application before its own 3-way handshake completes.

If a client uses TCP Fast Open cookies on both the parallel connection attempts of a dual handshake, an Upgraded Server will deliver the TCP Payload to the application twice before the client aborts the Ordinary Connection. This is not a problem, because [I-D.ietf-tcpm-fastopen] requires that TFO is only used for applications that are robust to duplicate requests.

2.3.1.2. Option Alignment

If the end of the last Inner TCP Option does not align on a 4-octet boundary, the sender MUST append sufficient no-op TCP options. On a SYN=1 segment, the end of the Prefix TCP Options MUST be similarly aligned.

If a block-mode transformation (e.g. compression or encryption) is being used, the sender might have to add some padding options to align the end of the Inner Options with the end of a block. Any future encryption specification will need to carefully define this padding in order not to weaken the cipher.

2.3.1.3. Sequence Space Coverage

TCP's sequence number and acknowledgement number space MUST include all the TCP Data, i.e. the InSpace Option, any Inner Options, and any magic number as well as the TCP Payload. Similarly, the sender MUST NOT transmit any form of TCP Data unless the advertised receive window is sufficient. These rules have significant implications, which are discussed in Section 3.2.4.

2.3.1.4. Presence or Absence of Payload

Whenever the sender includes non-zero user-data payload in a segment, it MUST also include an InSpace Option, whether or not there are any Inner Options.

If the sender includes no user-data in a segment (e.g. pure ACKs, RSTs) it MAY include an InSpace Option but it does not have to. {ToDo: Consider whether there is any reason to preclude Inner Options on a RST, FIN or FIN-ACK.}

Once a sender has included the InSpace Option and possibly other Inner Options on a segment with no TCP Payload, while it has no further user-data to send it SHOULD NOT repeat the same set of control options on subsequent segments. Thus, in a sequence of pure ACKs, any particular set of Inner Options will only appear once, and other pure ACKs will be empty. The only envisaged exception to this rule would be infrequent repetition (i.e. tens of minutes to hours) of the same control options, which might be necessary to provide a heartbeat or keep-alive capability.

2.3.2. Reading Inner TCP Options

The rules for reading Inner TCP Options are divided between the following two subsections, depending on whether SYN=1 or SYN=0.

2.3.2.1. Reading Inner TCP Options (SYN=1)

This subsection applies when TCP receives a segment with SYN=1, i.e. when the server receives a SYN or the client receives a SYN/ACK.

Before processing any TCP options, unless the size of the TCP Data is less than 8 octets, an Upgraded Receiver MUST determine whether the segment is an Upgraded Segment by checking that all the following conditions apply:

- The first 4 octets of the segment match Magic Number A;
- The value of the Length field of the InSpace Option is 2;
- The value of Magic Number B in the InSpace Option is correct;
- The value of the Sent Payload Size matches the size of the TCP Payload.

If all these conditions pass, the receiver MAY walk the sequence of Inner TCP Options, using the length of each to check that the sum of their lengths equals InOO. The receiver then concludes that the received segment is an Upgraded Segment.

The receiver then processes the TCP Options in the following order:

1. Any Prefix TCP options (PrefixOpts in Figure 2)
2. Any Outer TCP options (OuterOpts in Figure 2);
3. Any Suffix TCP options (SuffixOpts in Figure 2)

The receiver removes the magic number, the InSpace Option and each TCP Option from the TCP Data as it processes each. This frees up receive buffer, so the receiver increases its local value of the receive window accordingly. Once only the TCP Payload remains, the receiver holds it ready to pass to the application. It then returns the appropriate Upgraded Acknowledgement to progress the dual handshake (see Section 2.1.1).

If any of the above tests to find the InSpace Option fails:

1. the receiver concludes that the received segment is an Ordinary Segment. It MUST then proceed by processing any Outer TCP options in the TCP Header in the normal order (OuterOpts in Figure 2).

2. If some previous control message causes the TCP receiver to alter the TCP Data (e.g. decompression, decryption), it reruns the above tests to check if the altered TCP Data now looks like an Upgraded Segment.

3. If it finds an InSpace Option, it suspends processing the Outer TCP Options and instead processes and removes TCP Options in the following order:

   1. Any Prefix Inner Options;

   2. Any remaining Outer TCP Options;

   3. Any Suffix Inner Options.

4. If it does not find an InSpace Option, it continues processing the remaining Outer TCP Options as normal.

For the avoidance of doubt the above rules imply that, as long as an InSpace Option has not been found in the segment, the receiver might rerun the tests for it multiple times if multiple Outer TCP Options alter the TCP Data. However, once the receiver has found an InSpace Option, it MUST NOT rerun the tests for an Upgraded Segment in the same segment.

If the receiver has not found an InSpace Option after processing all the Outer Options, it returns the appropriate Ordinary Acknowledgement to progress the dual handshake (see Section 2.1.1). As normal, it holds any TCP Payload ready to pass to the application.

2.3.2.2. Reading Inner TCP Options (SYN=0)

This subsection applies once the TCP connection has successfully negotiated to use the upgraded InSpace structure.

As each segment with SYN=0 arrives, the receiver immediately processes any Outer TCP options.

As the receiver buffers TCP Data, it uses TCP’s regular mechanisms to fill any gaps due to reordering or loss so that it can work its way along the ordered byte-stream. As the receiver encounters each set of Inner Options, it MUST process them in the order they were sent, as illustrated in Figure 4a) in Section 3.2.4. The receiver MUST remove the InSpace Option and Inner TCP Options from the TCP Data as it processes them, adding to the receive window accordingly. Once only the TCP Payload remains the receiver passes it to the application.

It uses each InSpace Option to calculate the extent of the associated Inner Options (using InOO), and the amount of payload data before the next InSpace Option (using Sent Payload Size). The receiver MUST NOT locate InSpace Options by assuming there is one at the start of the TCP Data in every segment, because resegmentation might invalidate this assumption.

Therefore, the receiver processes the Inner Options in the order they were sent, which is not necessarily the order in which they are received. And if an Inner Option applies to the data stream, the receiver applies it at the point in the data stream where the sender inserted it. As a consequence, the receiver always processes the Inner Options after the Outer Options.

The Inner Options are deliberately placed within the byte-stream so that the sender can transform them along with the payload data, e.g. to compress or encrypt them. A previous control message might have required the TCP receiver to alter the byte-stream before passing it to the application, e.g. decompression or decryption. If so, the TCP receiver applies transformations progressively, to one sent segment at a time, in the following order:

1. The receiver MUST apply any transformations to the byte-stream up to the end of the next set of Inner Options, i.e. over the extent of the next Sent Payload Size, InSpace Option and any Inner Options.

2. The receiver MUST then process and remove the InSpace Option and any Inner Options (which might change the way it transforms the next segment, e.g. a rekey option).

3. Having established the extent of the next sent segment, The receiver returns to step 1.

2.3.3. Forwarding Inner TCP Options

Middleboxes exist that process some aspects of the TCP Header. Although the present specification defines a new location for Inner TCP Options beyond the Data Offset, this is intended for the exclusive use of the destination TCP implementation. Therefore:

- A middlebox MUST treat any octets beyond the Data Offset as immutable user-data. Legacy Middleboxes already do not expect to find options beyond the Data Offset anyway.

- A middlebox MUST NOT defer data in a segment with SYN=1 to a subsequent segment.

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 off-the-shelf TCP implementation. Therefore, a general-purpose TCP that implements the present specification will need a configuration switch to disable any search for options beyond the Data Offset and to enable immediate forwarding of data in a SYN.

2.4. Exceptions

{ToDo: Define behaviour of forwarding or receiving nodes if the structure or format of an Upgraded Segment is not as specified.}

If an Upgraded TCP Receiver receives an InSpace Option with a Length it does not recognise as valid, it MUST drop the packet and acknowledge the octets up to the start of the unrecognised option.

Values of Sent Payload Size greater than 2^16 - 25 (=65,511) octets in a regular (non-jumbo) InSpace Option MUST be treated as the distance to the next InSpace option, but they MUST NOT be taken as indicative of the size of the TCP Payload when it was sent. This is because the TCP Payload in a regular IPv6 packet cannot be greater than (2^16 -1 - 20 - 4) octets (given the minimum TCP header is 20 octets and the minimum InSpace Option is 4 octets). A Sent Payload Size of 0xFFFF octets MAY be used to minimise the occurrence of empty InSpace options without permanently disabling the Inner Space protocol for the rest of the connection.

If the size of the payload is greater than 65,511 octets, the sender MUST use a Jumbo InSpace Option (Appendix A.3).

2.5. SYN Flood Protection

An implementation of the Inner Space protocol MUST support the EchoCookie TCP option [I-D.briscoe-tcpm-echo-cookie]. To indicate its support for EchoCookie, an Ordinary Client would send an empty EchoCookie TCP option on the SYN. Support for the Inner Space protocol makes this redundant. Therefore an Inner Space client MUST NOT send an empty EchoCookie TCP option on a SYN-U.

The EchoCookie TCP option replaces the SYN Cookie mechanism [RFC4987], which only has sufficient space to hold the result of one TCP option negotiation (the MSS), and then only a subset of the possible values (see the discussion under Security Considerations Section 6).

3. Design Rationale

This section is informative, not normative.

3.1. Dual Handshake and Migration to Single Handshake

In traditional [RFC0793] TCP, the space for options is limited to 40B by the maximum possible Data Offset. Before a TCP sender places options beyond that, it has to be sure that the receiver will understand the upgraded protocol, otherwise it will confuse and potentially crash the application by passing it TCP options as if they were payload data.

The Dual Handshake (Section 2.1.1) ensures that a Legacy TCP Server will never pass on TCP options as if they were user-data. If a SYN carries TCP Data, a TCP server typically holds it back from the application until the 3-way handshake completes. This gives the client the opportunity to abort the Upgraded Connection if the response from the server shows it does not recognise an Upgraded SYN.

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 option 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 an Ordinary 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 an Ordinary 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 IPv6 then IPv4 [RFC6555];

- o and trying SCTP and TCP in parallel [I-D.wing-tsvwg-happy-eyeballs-sctp];
- o and trying ECN and non-ECN in parallel;
- o 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.

3.2. In-Band Inner Option Space

3.2.1. Non-Deterministic Magic Number Approach

This section justifies the magic number approach by contrasting it with a more 'conventional' approach. A conventional approach would use a regular (Outer) TCP option to point to the dividing line within the TCP Data between the extra Inner Options and the TCP Payload.

This 'conventional' approach cannot provide extra option space over a path on which a middlebox strips TCP options that it does not recognise. [Honda11] quantifies the prevalence of such paths. It reports on experiments conducted in 2010-2011 that found unknown options were stripped from the SYN-SYN/ACK exchange on 14% of paths to port 80 (HTTP), 6% of paths to port 443 (HTTPS) and 4% of paths to port 34343 (unassigned). Further analysis found that the option-stripping middleboxes fell into two main categories:

- o about a quarter appeared to actively remove options that they did not recognise (perhaps assuming they might be indicative of an attack?);
- o the rest were some type of higher layer proxy that split the TCP connection, unwittingly failing to pass unknown options between the two connections.

In contrast, the magic number approach ensures that not only are the Inner Options tucked away beyond the Data Offset, but the option that gives the extent of the Inner Options is also beyond the Data Offset (see Section 2.2.1). This ensures that all the TCP Headers and options up to the Data Offset are completely indistinguishable from an Ordinary Segment. It is very unusual for a middlebox not to forward TCP Data unchanged, so it will be highly likely (but not certain--see Appendix A.2.4) to forward the extra Inner Options.

The downside of the magic number approach is that it is slightly non-deterministic, quantified as follows:

- The probability that an Upgraded SYN=1 segment will be mistaken for an Ordinary Segment is precisely zero.

- In the currently common case of a SYN with zero payload, the probability that it will be mistaken for an Upgraded Segment is also precisely zero.

- However, there will be a very small probability (roughly 2^{-66} or 1 in 74 billion billion (74 * 10^18)) that payload data in an Ordinary SYN=1 segment could be mistaken for an Upgraded SYN or SYN/ACK, if it happens to contain a pattern in exactly the right place that matches the correct Sent Payload Size, Length and Magic Numbers of an InSpace Option. {ToDo: Estimate how often a collision will occur globally. Rough estimate: 1 connection collision globally every 40 years.}

The above probability is based on the assumptions that:

- the magic numbers will be chosen randomly (in reality they will not--for instance, a magic number that looked just like the start of an HTTP connection would be rejected)

- data at the start of Ordinary SYN=1 segments is random (in reality it is not--the first few bytes of most payloads are very predictable).

Therefore even though 2^{-66} is a vanishingly small probability, the actual probability of a collision will be much lower.

If a collision does occur, it will result in TCP removing a number of 32-bit words of data from the start of a byte-stream before passing it to the application.

3.2.2. Non-Goal: Security Middlebox Evasion

The purpose of locating control options within the TCP Data is not to evade security. Security middleboxes can be expected to evolve to examine control options in the new inner location. Instead, the purpose is to traverse middleboxes that block new TCP options unintentionally--as a side effect of their main purpose--merely because their designers were too careless to consider that TCP might evolve. This category of middleboxes tends to forward the TCP Payload unaltered.

By sitting within the TCP Data, the Inner Space protocol should traverse enough existing middleboxes to reach critical mass and prove itself useful. In turn, this will open an opportunity to introduce integrity protection for the TCP Data (which includes Inner Options). Whereas today, no operating system would introduce integrity protection of Outer TCP options, because in too many cases it would fail and abort the connection. Once the integrity of Inner Options is protected, it will raise the stakes. Any attempt to meddle with control options within the TCP Data will not just close off the theoretical potential benefit of a protocol advance that no-one knows they want yet; it will fail integrity checks and therefore completely break any communication. It is unlikely that a network operator will buy a middlebox that does that.

Then middlebox designers will be on the back foot. To completely block communications they will need a sound justification. If they block an attack, that will be fine. But if they want to block everything abnormal, they will have to block the whole communication, or nothing. So the operator will want to choose middlebox vendors who take much more care to ensure their policies track the latest protocol advances--to avoid costly support calls.

3.2.3. Avoiding the Start of the First Two Segments

Some middleboxes discard a segment sent to a well-known port (particularly port 80) if the TCP Data does not conform to the expected app-layer protocol (particularly HTTP). Often such middleboxes only parse the start of the app-layer header (e.g. Web filters only continue until they find the URL being accessed, or DPI boxes only continue until they have identified the application-layer protocol).

The segment structure defined in Section 2.2.1 would not traverse such middleboxes. An alternative segment structure that avoids the start of the first two segments in each direction is defined in Appendix A.4. It is not mandatory to implement in the present specification. However, it is hoped that it will be included in some experimental implementations so that it can be decided whether it is worth making mandatory.

3.2.4. Control Options Within Data Sequence Space

Including Inner Options within TCP’s sequence space gives the sender a simple way to ensure that control options will be delivered reliably and in order to the remote TCP, even if the control options are on segments without user-data. By using TCP’s existing stream delivery mechanisms, it adds no extra protocol processing, no extra packets and no extra bits.

The sender can even choose to place control options on a segment without user-data, e.g. to reliably re-key TCP-level encryption on a connection currently sending no data in one direction. The sender can even add an InSpace Option without further Inner Options. Then it can ensure that the segment will automatically be delivered reliably and in order to the remote TCP, even though it carries no user-data or other TCP control options, e.g. for a test probe, a tail-loss probe or a keep-alive.

Figure 4a) illustrates control options arriving reliably and in order at the receiving TCP stack in comparison with the traditional approach shown in Figure 4b), in which control options are outside the sequence space. In the traditional approach, during a period when the remote TCP is sending no user-data, the local TCP may receive control options E, B and D without ever knowing that they are out of order, and without ever knowing that C is missing.

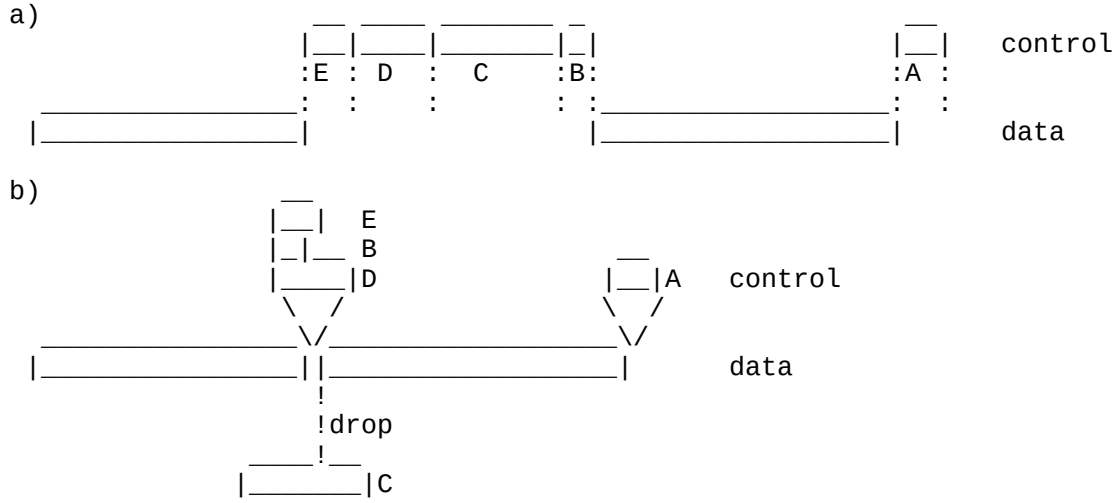


Figure 4: Control options a) inside vs. b) outside TCP sequence space‘

By including Inner Options within the sequence space, each control option is automatically bound to the start of a particular byte in the data stream, which makes it easy to switch behaviour at a specific point mid-stream (e.g. re-keying or switching to a different control mode). With traditional TCP options, a bespoke reliable and ordered binding to the data stream would have to be developed for each TCP option that needs this capability (e.g. co-ordinating use of new keys in TCP-AO [RFC5925] or tcpcrypt [I-D.bittau-tcpinc]).

Including Inner Options in sequence also allows the receiver to tell the sender the exact point at which it encountered an unrecognised TCP option using only TCP's pre-existing byte-granularity acknowledgement scheme.

Middleboxes exist that rewrite TCP sequence and acknowledgement numbers, and they also rewrite options that refer to sequence numbers (at least those known when the middlebox was produced, such as SACK, but not any introduced afterwards). If Inner Options were not included in sequence, the number of bytes beyond the TCP Data Offset in each segment would not match the sequence number increment between segments. Then, such middleboxes could unintentionally corrupt the user-data and options by 'normalising' sequence or acknowledgement numbering. Fortunately, including Inner Options in sequence improves robustness against such middleboxes.

3.2.5. Rationale for the Sent Payload Size Field

A middlebox that splits a TCP connection can coalesce and/or divide the original segments. Segmentation offload hardware introduces similar resegmentation. Inclusion of the Sent Payload Size field in the InSpace Option makes the scheme robust against such resegmentation.

The Sent Payload Size is not strictly necessary on a SYN (SYN=1, ACK=0) because a SYN is never resegmented. However, for simplicity, the layout for a SYN is made the same as for a SYN/ACK. This future-proofs the protocol against the possibility that SYNs might be resegmented in future. And it makes it easy to introduce the alternative segment structure of Appendix A.4 if it is needed.

3.3. Rationale for the InSpace Option Format

The format of the InSpace Option (Figure 3) does not necessarily have to comply with the RFC 793 format for TCP options, because it is not intended to ever appear in a sequence of TCP options. In particular, it does not need an Option Kind, because the option is always in a known location. In effect the magic number serves as a multi-octet Option Kind for the first InSpace Option, and the location of each subsequent options is always known as an offset from the previous one, using InOO and Sent Payload Size fields.

Other aspects of the layout are justified as follows:

Length: Whatever the size of the InSpace Option, the right-hand edge of the Length field is always located 4 octets from the start of the option, so that the receiver can find it to determine the layout of the rest of the option. The option is always a multiple

of 4 octets long, so that any subsequent Inner TCP Options comply with TCP's option alignment requirements.

Sent Payload Size: This field is 16 bits wide, which is reasonable given segment size cannot exceed the limits set by the Total Length field in the IPv4 header and the Payload Length field in the IPv6 header, both of which are 16 bits wide.

If the sender were to use a jumbogram [RFC2675], it could use the Jumbo InSpace Option defined in Appendix A.3, which offers a 32-bit Sent Payload Size field. The Jumbo InSpace Option is not mandatory to implement for the present experimental specification. Even if it is implemented, it is only defined when SYN=0, given use of a jumbogram for a SYN or SYN/ACK would significantly exceed other limits that TCP sets for these segments.

InSpace Options Offset The 14-bit field is in units of 4-octet words, in order to restrict Inner Options to no less than the size of a maximum sized segment (given 4 * 2^14 = 2^16 octets).

When SYN=1 the layout of the InSpace Option is extended to include:

Suffix Options Offset: The SOO field is the same 14-bit width as the InOO field, and for the same reason. Both the SOO and InOO fields are aligned 2 bits to the left of a word boundary so that they can be used directly in units of octets by masking out the 2-bit field to the right.

Magic Number B: The 32-bit size of Magic Number A is not enough to reduce the probability of mistaking the start of an Ordinary SYN Payload for the start of the Inner Space protocol. A 64-bit magic number could have been provided by using the next 4-octet word, but this would be unnecessarily large. Therefore, when SYN=1, 16 more bits of magic number are provided within the InSpace Option. Otherwise, these 16-bits would only have to be used for padding to align with the next 4-octet word boundary anyway.

3.4. Protocol Overhead

The overhead of the Inner Space protocol is quantified as follows:

Dual Handshake:

Latency:

Upgraded Server : zero;

Legacy Server: worst latency of the dual handshakes.

Connection Rate: The typical connection rate will inflate by P*D, where:

P [0-100%] is the proportion of connections that use extra option space;

D [0-100%] is the proportion of these that use a dual handshake (the remainder use a single handshake, e.g. by caching knowledge of upgraded servers).

For example, if P=80% and D=10%, the connection rate will inflate by 8%. P is difficult to predict. D is likely to be small, and in the longer term it should reduce to the proportion of connections to remaining legacy servers, which are likely to be the less frequently accessed ones. In the worst case if both P & D are 100%, the maximum that the connection rate can inflate by is 100% (i.e. to twice present levels).

Connection State: Connection state on servers and middleboxes will inflate by P*D/R, where

R is the average hold time of connection state measured in round trip times

This is because a server or middlebox only holds dual connection state for one round trip, until the RST on one of the two connections. For example, keeping P & D as they were in the above example, if R = 3 round trips {ToDo: TBA}, connection state would inflate by 2.7%. In the longer term, any extra connection state would be focused on legacy servers, with none on upgraded servers. Therefore, if memory for dual handshake flow state was a problem, upgrading the server to support the Inner Space protocol would solve the problem.

Network Traffic: The network traffic overhead is 2*H*P*D/J counting in bytes or 2*P*D/K counting in packets, where

H is 88B for IPv4 or 108B for IPv6 (assuming the Ordinary SYN and SYN/ACK have a TCP header packed to the maximum of 60B with TCP options, they have no TCP payload, their IP headers have no extensions and the InSpace Option in the SYN-U and SYN/ACK-U is 8B);

J is the average number of bytes per TCP connection (in both directions)

K is the average number of packets per TCP connection (in both directions);

For example, keeping and P & D as they were in the above example, if J = 50KiB for IPv4 and K = 70 packets (ToDo: TBA), traffic overhead would be 0.03% counting in bytes or 0.2% counting in packets.

Processing: {ToDo: Implementation tests}

InSpace Option on every non-empty SYN=0 segment:

Network Traffic: The traffic overhead is P*Q*4/F, where

Q is the proportion of Inner Space connections that leave the protocol enabled after the initial handshake;

F is the average frame size in bytes (assuming one segment per frame).

This is because the InSpace option adds 4B per segment. For example, keeping P as it was in the above example and taking Q=10% and F=750B, the traffic overhead is 0.04%. It is as difficult to predict Q as it is to predict P.

Processing: {ToDo: Implementation tests}

4. Interaction with Pre-Existing TCP Implementations

4.1. Compatibility with Pre-Existing TCP Variants

A TCP option MUST by default only be used as an Outer Option, unless it is explicitly specified that it can (or must) be used as an Inner Option. The following list of pre-existing TCP options can be located as Inner Options:

- Maximum Segment Size (MSS) [RFC0793];
- SACK-ok [RFC2018];
- Window Scale [RFC7323];
- Multipath TCP [RFC6824], except the Data ACK part of the Data Sequence Signal (DSS) option;
- TCP Fast Open [I-D.ietf-tcpm-fastopen];
- The tcpcrypt CRYPT option [I-D.bittau-tcpinc].

The following MUST NOT be located as Inner Options:

- Timestamp [RFC7323];
- SACK [RFC2018];
- The Data ACK part of the DSS option of Multipath TCP [RFC6824];
- TCP-AO [RFC5925];
- The tcpcrypt MAC option [I-D.bittau-tcpinc] as long as it covers the TCP header.

{ToDo: The above list is not authoritative. Many of the above schemes involve multiple different types of TCP option, and all the types need to be separately assessed.}

The Inner Space protocol supports TCP Fast Open, by constraining the client to obey the rules in Section 2.3.1.1).

All the sub-types of the MPTCP option [RFC6824] except one could be located as Inner Options. That is, MP_CAPABLE, MP_JOIN, ADD_ADDR(2), REMOVE_ADDR, MP_PRIO, MP_FAIL, MP_FASTCLOSE. The Data Sequence Signal (DSS) of MPTCP consists of four separable parts: i) the Data ACK; ii) the mapping between the Data Sequence Number and the Subflow Sequence Number over a Data-Level Length; iii) the Checksum; and iv) the DATA_FIN flag. If MPTCP were re-factored to take advantage of the Inner Space protocol, all these parts except the Data ACK could be located as Inner Options (the Checksum would not be necessary).

The MPTCP Data ACK has to remain as an Outer Option otherwise there would be a risk of flow control deadlock, as pointed out in [Raiciu12]. For instance, a Web client might pipeline multiple requests that fill a Web server’s receive buffer, while the Web server might be busy sending a large response to the first request before it reads the second request. If the Data ACK were an Inner Option, the Web client would have to stop acknowledging the first response from the server (due to lack of receive window). Then the server would not be able to move on to the next request--a classic deadlock.

The TCP-AO has to be located as an Outer Option to prevent the possibility of flow-control deadlock (because it would consume receive window on pure ACKs).

All sub-options of the tcpcrypt CRYPT option could be located as Inner Options. However, as long as the tcpcrypt MAC option covers

the TCP header and Outer Options, it has to be located as an Outer Option for the same deadlock reason as TCP-AO.

An Upgraded Server can support SYN Cookies [RFC4987] for Ordinary Connections. For Upgraded Connections Section 2.5 defines a new EchoCookie TCP option that is a prerequisite for InSpace implementations, and provides sufficient space for the more extensive connection state requirements of an InSpace server.

{ToDo: TCP States and Transitions, Connectionless Resets, ICMP Handling, Forward-Compatibility.}

4.2. Interaction with Middleboxes

The interaction with the assumptions about TCP made by middleboxes is covered extensively elsewhere:

- Section 2.3.3 specifies forwarding behaviour for Inner Options;
- The following sections explain the Inner Space protocol approach to middlebox traversal:
  - Section 3.2.1 justifies the magic number approach;
  - Section 3.2.2 explains why the protocol will remain robust as middlboxes evolve;
  - Section 3.2.4 justifies including Inner Options in sequence;
  - Section 3.2.5) explains how the protocol will remain robust to resegmentation.

4.3. Interaction with the Pre-Existing TCP API

An aim of the Inner Space protocol is for legacy applications to continue to just work without modification. Therefore it is expected that the dual handshaking logic and any maintenance of a cached white-list of servers that support the Inner Space protocol will be implemented beneath the well-known socket interface.

Inner Space implementations will need to comply with the following behaviours to ensure that legacy applications continue to receive predictable behaviour from the socket interface:

Querying local port (TCP client): If an application calls "getsockname()" while the TCP client behind the socket is engaged in a dual TCP handshake, the call SHOULD block until the local TCP

has aborted one of the connections so it knows which of the two ports will continue to be used.

Binding to an explicit port: If an application specifies that it wants the TCP client to use a specific port, the Inner Space capability MUST be disabled, because the dual handshake has to try two ports. Use of a specific port might be necessary, for example in a port-testing application or if the application wants to explicitly control all the handshaking logic of the Inner Space protocol itself.

Logging: The dual handshake will show up as a specific signature in logs of network activity. Log formats might not be able to record two local ports against one socket, so logs might contain unexpected or erroneous data. Even if logs correctly track both connection attempts, log analysis software might not expect to see one socket attempt to use two different ports. {ToDo: All this needs to be turned into a predictability requirement.}

Note that Inner Space has no impact on queries for the remote port from a TCP server. If an application calls "getpeername()" while the TCP server behind the socket is (unwittingly) engaged in a dual handshake, it will return the port of the remote client, even though this connection might subsequently be aborted. This is because a TCP server is not aware of whether it is part of a dual handshake.

It would be appropriate to enable the Inner Space protocol on a per-host or per-user basis. The necessary configuration switch does not need to be standardised, but it might allow the following three states:

Enabled: The stack will enable Inner Space on any TCP connection that that needs Inner Space for its TCP options. The stack might still disable the Inner Space protocol autonomously after the initial handshake if it is not needed.

Forwarding: The Forwarding mode is for TCP implementations on middleboxes that implement split TCP connections, as discussed in Section 2.3.3. Forwarding mode is similar to Disabled, except it forwards data in SYN without deferring it until the incoming connection is established.

Disabled: Inner Space is not enabled by default on any connections, except those that specifically request it.

The socket API might also need to be extended for future applications that want to control the Inner Space protocol explicitly. Experience

will determine the best API, so these ideas are merely informational suggestions at this stage:

Enabling/disabling Inner Space: As well as the above per-host or per-user switches, the extended API might need to allow an application to disable Inner Options on a per-socket basis (e.g. for testing). A socket might need to be opened in one of three possible Inner Space modes: i) Enabled; ii) Enabled initially but can be disabled autonomously by the stack if redundant; iii) Enabled initially, then disables itself after the SYN/ACK; and iv) Disabled. It also ought to be possible for an application to disable Inner Options on-demand mid-connection.

Querying support for Inner Space: An application might need to be able to determine whether the host supports Inner Space and in which mode it is enabled on a particular socket. For instance, an application might need to choose different socket options depending on whether Inner Space is enabled to make the necessary space available.

Latency vs Efficiency: A socket that prefers efficient use of connection state over latency might use the optional explicit variant of the dual handshake (Appendix B). It is unlikely that a new option specific to Inner Space would be needed to express this preference, as many operating systems already offer a similar socket option.

Logging: Log formats and log analysis software might need to be extended to distinguish between the deliberate RST within the dual handshake and an unexpected connection RST.

5. IANA Considerations

This specification requires IANA to allocate values from the TCP Option Kind name-space against the following names:

- "Inner Option Space Upgraded (InSpaceU)"
- "Inner Option Space Ordinary (InSpaceO)"
- "ModeSwitch"

Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and respective Experiment IDs:

- 0xUUUU (16 bits);
- 0xOOOO (16 bits);

o 0xMMMM (16 bits);

{ToDo: Values TBA and register them with IANA} then migrate to the assigned option after allocation.

6. Security Considerations

Certain cryptographic functions have different coverage rules for the TCP Header and TCP Payload. Placing some TCP options beyond the Data Offset 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.

A malicious host can send bogus SYN segments with a spoofed source IP address (a SYN flood attack). The Inner Space protocol does not alter the feasibility of this attack. However, the extra space for TCP options on a SYN allows the attacker to include more TCP options on a SYN than before, so it can make a server do more option processing before replying with a SYN/ACK. To mitigate this problme, a server under stress could deprioritise SYNs with longer option fields to focus its resources on SYNs that require less processing.

Each SYN in a SYN flood attack causes a TCP server to consume memory. The Inner Space protocol allows a potentially large amount of TCP option state to be negotiated during the SYN exchange, which could exhaust the TCP server’s memory. The EchoCookie TCP option (see Section 2.5) allows the server to place this state in a cookie and send it on the SYN/ACK to the purported address of the client--rather than hold it in memory.

Then, as long as the client returns the cookie on the acknowledgement and the server verifies it, the server can recover its full record of all the TCP options it negotiated and continue the connection without delay. On the other hand, the server’s responses to SYNs from spoofed addresses will scatter to those spoofed addresses and the server will not have consumed any memory while waiting in vain for them to reply. See the Security Considerations in [I-D.briscoe-tcpm-echo-cookie] for how the EchoCookie facility protects against reflection and amplification attacks.

7. 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 within a higher layer, and code its location into upgraded implementations of the lower layer, was originally articulated by Rob Hancock. {ToDo: Ref?} The following

people provided useful review comments: Joe Touch, Yuchung Cheng, John Leslie, Mirja Kuehlewind, Andrew Yourtchenko, Costin Raiciu, Marcelo Bagnulo Braun, Julian Chesterfield and Jaime Garcia.

Bob Briscoe's contribution is part-funded by the European Community under its Seventh Framework Programme through the Trilogy 2 project (ICT-317756) and the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the author.

8. References

8.1. Normative References

[I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-10 (work in progress), September 2014.

[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.

8.2. Informative References

[Honda11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it Still Possible to Extend TCP?", Proc. ACM Internet Measurement Conference (IMC'11) 181--192, November 2011.

[I-D.bittau-tcpinc]
Bittau, A., Boneh, D., Hamburg, M., Handley, M., Mazieres, D., and Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-bittau-tcpinc-01 (work in progress), July 2014.

[I-D.briscoe-tcpm-echo-cookie]
Briscoe, B., "The Echo Cookie TCP Option", draft-briscoe-tcpm-echo-cookie-00 (work in progress), October 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.

[Iyengar10]
Iyengar, J., Ford, B., Ailawadi, D., Amin, S., Nowlan, M., Tiwari, N., and J. Wise, "Minion--an All-Terrain Packet Packhorse to Jump-Start Stalled Internet Transports", Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDnet'10) , November 2010.

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

[RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[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.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", RFC 7323, September 2014.

[Raiciu12]
Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", Proc. USENIX Symposium on Networked Systems Design and Implementation , April 2012.

Appendix A. Protocol Extension Specifications

This appendix specifies protocol extensions that are OPTIONAL while the specification is experimental. If an implementation includes an extension, this section gives normative specification requirements.

However, if the extension is not implemented, the normative requirements can be ignored.

{Temporary note: The IETF may wish to consider making some of these extensions mandatory to implement if early testing shows they are useful or even necessary. Or it may wish to make at least the receiving side mandatory to implement to ensure that two-ended experiments are more feasible.}

A.1. Disabling InSpace and Generic Connection Mode Switching

This appendix is normative. It is separated from the body of the specification because it is OPTIONAL to implement while the Inner Space protocol is experimental. It defines the new ModeSwitch TCP option illustrated in Figure 5. This option provides a facility to disable the Inner Space protocol for the remainder of a connection. It also provides a general-purpose facility for a TCP connection to co-ordinate between the endpoints before switching into a yet-to-be-defined mode.

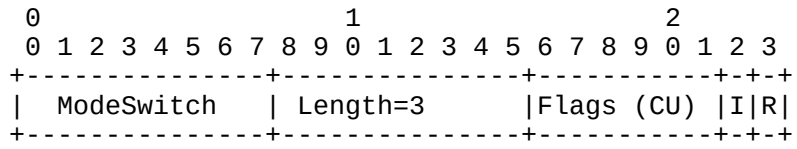


Figure 5: The ModeSwitch TCP Option

The Option Kind is ModeSwitch, the value of which is to be allocated by IANA {ToDo: Value TBA}. ModeSwitch MUST be used only as an Inner Option, because it uses the reliable ordered delivery property of Inner Options. Therefore implementation of the Inner Space protocol is REQUIRED for an implementation of ModeSwitch. Nonetheless, ModeSwitch is a generic facility for switching a connection between yet-to-be-defined modes that do not have to relate to extra option space.

The sender MUST set the option Length to 3 (octets). The Length field MUST be forwarded unchanged by other nodes, even if its value is different.

The Flags field is available for defining modes of the connection. Only two connection modes are currently defined. The first 6 bits of the Flags field are Currently Unused (CU) and the sender MUST set them to zero. The CU flags MUST be ignored and forwarded unchanged by other nodes, even if their value is non-zero.

The two 1-bit connection mode flags that are currently defined have the following meanings:

- R: Request flag if 1. Request mode is a special mode that allows the hosts to co-ordinate a change to any other mode(s);

- I: Inner Space mode: Enabled if 1, Disabled if 0.

The default Inner Space mode at the start of a connection is I=1, meaning Inner Space is in enabled mode.

The procedure for changing a mode or modes is as follows:

- The host that wants to change modes (the requester) sends a ModeSwitch message as an Inner Option with R=1 and with the other flag(s) set to the mode(s) it wants to change to. The requester does not change modes yet.

- The responder echoes the mode flag(s) it is willing to change to, with the request flag R=0.

- The half-connection from the responder changes to the mode(s) it confirms directly after the end of the segment that echoes its confirmation, i.e. after the last octet of the TCP Payload following the ModeSwitch option that echoes its confirmation. Therefore it sends the segment carrying the confirmation in the prior mode(s) of the connection.

- Once the requester receives the responder’s confirmation message, it re-echoes its confirmation of the responder’s confirmation, with the mode(s) set to those that both hosts agree on and R=0.

- The half-connection from the requester changes to the mode(s) it confirms directly after the end of the segment that re-echoes its confirmation. Therefore it sends the segment carrying the confirmation in the prior mode(s) of the connection.

- The responder can refuse a request to change into a mode in any one of three ways:

  * either implicitly by never confirming it;

  * or explicitly by sending a message with R=0 and the opposite mode;

  * or explicitly be sending a counter-request to switch to the opposite mode (that the connection is already in) with R=1.

The regular TCP sequence numbers and acknowledgement numbers of requests or confirmations can be used to disambiguate overlapping requests or responses.

Once a host switches to Disabled mode, it MUST NOT send any further InSpace Options. Therefore it can send no further Inner Options and it cannot switch back to Enabled mode for the rest of the connection.

To temporarily reduce InSpace overhead without permanently disabling the protocol, the sender can use a value of 0xFFFF in the Sent Payload Size (see Section 2.4).

A.2. Dual Handshake: The Explicit Variant

This appendix is normative. It is separated from the body of the specification because it is OPTIONAL to implement while the Inner Space protocol is experimental. It is not mandatory to implement because it will be more useful once the Inner Space protocol has become accepted widely enough that fewer middleboxes will discard SYN segments carrying this option (see Appendix B for when best to deploy it). It only works if both ends support it, but it can be deployed one end at a time, so there is no need for support in early experimental implementations.

{Temporary note: The choice between the explicit handshake in the present section or the handshake in Section 2.1.1 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 require a server to understand 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.}

This explicit dual handshake is similar to that in Section 2.1.1, except the SYN that the Upgraded Client sends on the Ordinary Connection is explicitly distinguishable from the SYN that would be sent by a Legacy Client. Then, if the server actually is an Upgraded Server, it can reset the Ordinary Connection itself, rather than creating connection state for at least a round trip until the client resets the connection.

For an explicit dual handshake, the TCP client still sends two alternative SYNs: a SYN-O 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 within the TCP Data (the SYN-U defined as before in Section 2.2.1).

Table 2 summarises the TCP 3-way handshake exchange for each of the two SYNs in the two right-hand columns, between an Upgraded TCP Client (the active opener) and either:

1. a Legacy Server, in the top half of the table (steps 2-4), or

2. an Upgraded Server, in the bottom half of the table (steps 2-4)

The table uses the same layout and symbols as Table 1, which has already been explained in Section 2.1.1.

		Ordinary Connection	Upgraded Connection
1	Upgraded Client	>SYN-O	>SYN-U
/\/\	/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\
2	Legacy Server	<SYN/ACK	<SYN/ACK
3a	Upgraded Client	Waits for response to both SYNs	
3b	"	>ACK	>RST
4		Cont...	
/\/\	/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\	/\/\/\/\/\/\/\/\
2	Upgraded Server	<RST	<SYN/ACK-U
3	Upgraded Client		>ACK
4			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-O with a RST. However, the client cannot rely on this behaviour, because a

middlebox might be stripping Outer TCP Options which would turn the SYN-O into a regular SYN before it reached 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 2.1.1.

The rules for processing TCP options are also unchanged from those in Section 2.3.

A.2.1. SYN-O Structure

The SYN-O is merely a SYN with an extra InSpaceO Outer TCP Option as shown in Figure 6. It merely identifies that the SYN is opening an Ordinary Connection, but explicitly identifies that the client supports the Inner Space protocol.

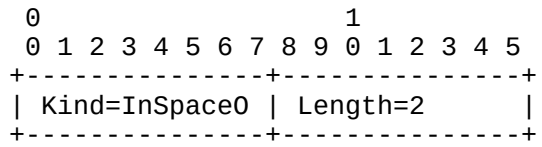


Figure 6: An InSpaceO TCP Option Flag

An InSpaceO TCP Option has Option Kind InSpaceO with value {ToDo: Value TBA} and MUST have Length = 2 octets.

To use this option, the client MUST place it with the Outer TCP Options. A Legacy Server will just ignore this TCP option, which is the normal behaviour for an option that TCP does not recognise [RFC0793].

A.2.2. Retransmission Behaviour - Explicit Variant

If the client receives a RST on one connection, but a short while after that {ToDo: 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 option space, in parallel to any retransmission, or instead of any retransmission, the client MAY send a SYN without any InSpace TCP Option, in case this is the cause of the black-hole. However, the presence of the RST implies that the SYN with the InSpaceO TCP Option (the SYN-O) 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-O or the SYN-U, it SHOULD solely retransmit one or the other, not both. If latency is more important than the extra TCP option space, it SHOULD send a SYN without an InSpaceO 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.

A.2.3. Corner Cases

There is a small but finite possibility that the Explicit Dual Handshake might encounter the cases below. The Implicit Handshake (Section 2.1.1) is robust to these possibilities, but the Explicit Handshake is not, unless the following additional rules are followed:

Both successful: This could occur if one load-sharing replica of a server is upgraded, while another is not. This could happen in either order but, in both cases, the client aborts the last connection to respond:

* The client completes the Ordinary 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 Inner TCP Options and one without. This could confuse the application.

* The client completes the Upgraded Connection after receiving a SYN/ACK-U, but then it receives a SYN/ACK in response to the SYN-O. In this case, the client MUST abort the connection it initiated with the SYN-O.

Both aborted: The client might receive a RST in response to its SYN-O, then an Ordinary SYN/ACK on its Upgraded Connection in response to its SYN-U. This could occur i) if a split connection middlebox actively forwards unknown options but holds back or discards data in a SYN; or ii) if one load-sharing replica of a server is upgraded, while another is not.

Whatever the likely cause, the client MUST still respond with a RST on its Upgraded Connection. Otherwise, its Inner TCP Options will be passed as user-data to the application by a Legacy Server.

If confronted with this scenario where both connections are aborted, the client will not be able to include extra options on a SYN, but it might still be able to set up a connection with extra option space on all the other segments in both directions using the approach in Appendix A.2.4. If that doesn’t work either, the

client's only recourse is to retry a new dual handshake on different source ports, or ultimately to fall-back to sending an Ordinary SYN.

A.2.4. Workround if Data in SYN is Blocked

If a path either holds back or discards data in a SYN-U, but there is evidence that the server is upgraded from a RST response to the SYN-O, the strategy below might at least allow a connection to use extra option space on all the segments except the SYN.

It is assumed that the symptoms described in the 'both aborted' case (Appendix A.2.3) have occurred, i.e. the server has responded to the SYN-O with a RST, but it has responded to the SYN-U with an Ordinary SYN/ACK not a SYN/ACK-U, so the client has had to RST the Upgraded Connection as well. In this case, the client SHOULD attempt the following (alternatively it MAY give up and fall back to opening an Ordinary TCP connection).

The client sends an 'Alternative SYN-U' by including an InSpaceU Outer TCP Option (Figure 7). This Alternative SYN-U merely flags that the client is attempting to open an Upgraded Connection. The client MUST NOT include any Inner Options or InSpace Option or Magic Number. If the previous aborted SYN/ACK-U acknowledged the data that the client sent within the original SYN-U, the client SHOULD resend the TCP Payload data in the Alternative SYN-U, otherwise it might as well defer it to the first data segment.

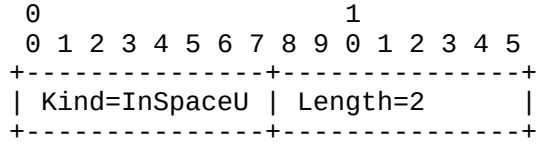


Figure 7: An InSpaceU Flag TCP option

An InSpaceU Flag TCP Option has Option Kind InSpaceU with value {ToDo: Value TBA} and MUST have Length = 2 octets.

To use this option, the client MUST place it with the Outer TCP Options. A Legacy Server will just ignore this TCP option, which is the normal behaviour for an option that TCP does not recognise [RFC0793]. Because the client has received a RST from the server in response to the SYN-O it can assume that the server is upgraded. So the client probably only needs to send a single Alternative SYN-U in this repeat attempt. Nonetheless, the RST might have been spurious.

Therefore the client MAY also send an Ordinary SYN in parallel, i.e. using the Implicit Dual Handshake (Section 2.1.1).

If an Upgraded Server receives a SYN carrying the InSpaceU option, it MUST continue the rest of the connection as if it had received a full SYN-U (Section 2.2), i.e. by processing any Outer Options in the SYN-U and responding with a SYN/ACK-U.

A.3. Jumbo InSpace TCP Option (only if SYN=0)

This appendix is normative. It is separated from the body of the specification because it is OPTIONAL to implement while the Inner Space protocol is experimental. In experimental implementations, it will be sufficient to implement the required behaviour for when the Length of a received InSpace Option is not recognised (Section 2.4).

If the IPv6 Jumbo extension header is used, the SentPayloadSize field will need to be 4 octets wide, not 2 octets. This section defines the format of the InSpace Option necessary to support jumbograms.

If sending a jumbogram, a sender MUST use the InSpace Option format defined in Figure 8. All the fields have the same meanings as defined in Section 2.2.2, except InOO and SentPayloadSize use more bits.

When reading a segment, the Jumbo InSpace Option could be present in a packet that is not a jumbogram (e.g. due to resegmentation). Therefore a receiver MUST use the Jumbo InSpace Option to work along the stream irrespective of whether arriving packets are jumbo sized or not.

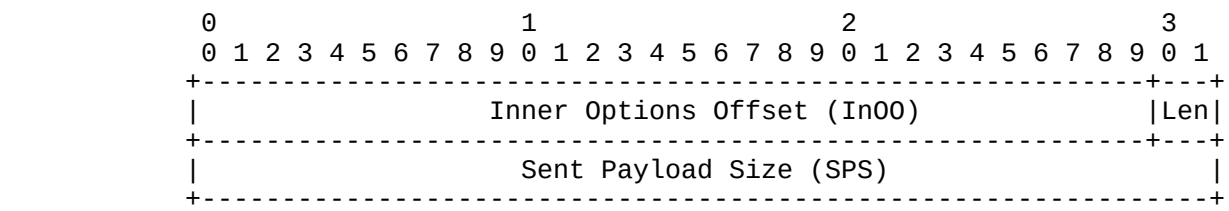


Figure 8: InSpace Option for a Jumbo Data-UNJH

A.4. Upgraded Segment Structure to Traverse DPI boxes

This appendix is normative. It is separated from the body of the specification because it is OPTIONAL to implement while the Inner Space protocol is experimental. If a receiver has implemented the Inner Space protocol but not this extension, no mechanism is provided for it to ask the sender to fall-back to the base Inner Space

protocol if it is sent a segment formatted according to this extension. However, it will at least fall-back naturally to regular TCP behaviour because of the dual handshake.

In experiments conducted between 2010 and 2011, [Honda11] reported that 7 of 142 paths (about 5%) blocked access to port 80 if the payload was not parsable as valid HTTP. This variant of the specification has been defined in case experiments prove that it significantly improves traversal of such deep packet inspection (DPI) boxes.

This variant starts the TCP Data with the expected app-layer headers on the first two segments in each direction:

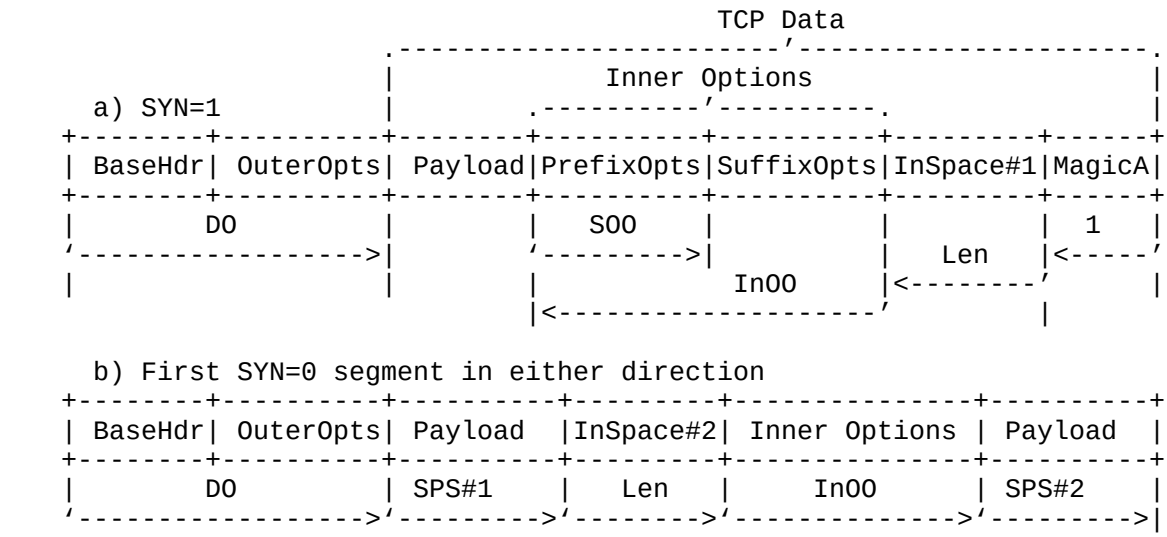
SYN=1: The structure in Figure 9a) is used on a SYN or SYN/ACK. The sender locates the 4-octet Magic Number A at the end of the segment. The sender right-aligns the 8-octet InSpace Option just before Magic Number A. Then it right-aligns the Inner Options against the InSpace Option, all after the end of the TCP Payload. The start of the Inner Options is therefore 4 * (InOO +3) octets before the end of the segment, where InOO is read from within the InSpace Option.

A receiver implementation will check whether Magic Number A is present at the end of the segment if it does not first find it at the start of the segment. Although the InnerOptions are located at the end of the TCP Payload, they are considered to be applied before the first octet of the TCP Payload.

SYN=0: The structure of the first non-SYN segment that contains any TCP Data is shown in Figure 9b).

The receiver will find the second InSpace Option (InSpace#2) located SPS#1 octets from the start of the segment, where SPS#1 is the value of Sent Payload Size that was read from the InSpace Option in the previous (SYN=1) segment that started the half-connection. Although the Inner Options are shifted, as for the first segment, they are still considered to be applied at the start of the TCP Data in this second segment.

From the second InSpace Option onwards, the structure of the stream reverts to that already defined in Section 2.2.1. So the value of Sent Payload Size (SPS#2) in the second InSpace Option (InSpace #2) defines the length of any remaining TCP Payload before the end of the first data segment, as shown.



All offsets are specified in 4-octet (32-bit) words, except SPS, which is in octets.

Figure 9: Segment Structures to Traverse DPI boxes (not to scale)

It is recognised that having to work from the end of the first segment makes processing more involved. Experimental implementation of this approach will determine whether the extra complexity improves DPI box traversal sufficiently to make it worthwhile.

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 2.1.1) starting with just an Ordinary SYN (no InSpaceO flag option) on the Ordinary Connection;

2. The explicit dual handshake (Appendix A.2) starting with a SYN-O (InSpaceO flag option) on the Ordinary 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-O (it is possible that some firewalls will discard packets with unrecognised TCP options {ToDo: ref?}). 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 Ordinary Connection

There is no need for the IETF to choose between these. If the specification 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.

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.

Option alignment following re-segmentation: If the byte-stream is resegmented (e.g. by a connection splitter), the TCP options within the stream will not necessarily align on 4-octet word boundaries within the new segments.

Ossifies reliable ordered delivery into TCP design: At present it is theoretically possible to implement a variant of TCP that provides partial reliability. Inner Space as it stands would prevent a future partial reliable TCP, but not if out-of-order delivery were added, as discussed below.

Ideally Outer Options in Inner: Ideally enable Outer Options to be located beyond the Data Offset: i) without consuming receive window ii) either without consuming sequence space or, if otherwise, must be robust to middlebox correction; iii) delivered immediately on reception, not in sent order. Could use the Minion

[Iyengar10] variant (or a similar variant) of the consistent overhead byte-stuffing (COBS) encoding.

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

A detailed version history can be accessed at <http://datatracker.ietf.org/doc/draft-briscoe-tcpm-inner-space/history/>

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

Technical changes:

* Corrected DO to 4 * DO (twice)
* Confirmed that receive window applies to Inner Options
* Generalised the cause of decryption/decompression from a previous TCP option to any previouis control message
* Added requirement for a middlebox not to defer data on SYN
* Latency of dual handshake is worst of two
* Completed "Interaction with Pre-Existing TCP Implementations" section, covering other TCP variants, TCP in middleboxes and the TCP API. Shifted some TCP options to Outer only, because of RWND deadlock problem
* Added two outstanding issues: i) ossifies reliable ordered delivery; ii) Ideally Outer in Inner.

Editorial changes:

* Removed section on Echo TCP option to a separate I-D that is mandatory to implement for inner-space, and shifted some SYN flood discussion in Security Considerations
* Clarifications throughout
* Acknowledged more review comments

From draft-briscoe-tcpm-syn-op-sis-02 to draft-briscoe-tcpm-inner-space-00:

The Inner Space protocol is a development of a proposal called the SynOpSis (Sister SYN options) protocol. Most of the elements of Inner Space were in SynOpSis, such as the implicit and explicit dual handshakes; the use of a magic number to flag the existence

of the option; the various header offsets; and the option processing rules.

The main technical differences are: Inner Space extends option space on any segment, not just the SYN; this advance requires the introduction of the Sent Payload Size field and a general rearrangement and simplification of the protocol format; the option processing rules have been extended to assure compatibility with TFO and one degree of recursion has been introduced to cater for encryption or compression of Inner Options; The Echo option has been added to provide a SYN-cookie-like capability. Also, the default protocol has been pared down to the bare bones and optional extensions relegated to appendices.

The main editorial differences are: The emphasis of the Abstract and Introduction has expanded from a focus on just extra space using the dual handshake to include much more comprehensive middlebox traversal. A comprehensive Design Rationale section has been added.

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/