TCPM WG
Internet-Draft
Intended status: Informational
Expires: September 26, 2012

A. Ramaiah
Cisco

March 25, 2012

TCP option space extension
draft-ananth-tcpm-tcpoptext-00.txt

Abstract

The document goals are as follows: Firstly, this document summarizes the motivations for extending TCP option space. Secondly, It tries to summarize the various known issues that needs to be taken into account while extending the TCP option space. Thirdly, it briefly provides a short summary of the various TCP option space proposals that has been proposed so far. Some additional proposals which includes variations to the existing proposals are also presented. The goal of this document is to rejuvenate the discussions on this topic and eventually to converge on a scheme for extending TCP option space.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 26, 2012.

Copyright Notice

Table of Contents

1. 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]

Middlebox : the middlebox could be a proxy like PEP (performance enhancing proxies), NAT device or a Firewall. Note : The term packet and segment is used interchangeably in this document.

2. Introduction

TCP [RFC0793] as part of its header includes TCP options which provides some special functionality that can be negotiated by both ends of the TCP connection. As the name implies, these aren't mandatory but must be supported by both ends in order to be active on a given TCP connection. Some examples of deployed TCP options include TCP MSS, Window Scale, Selective ACK, Timestamps ([RFC1323]). Many other TCP options have been proposed and have been in use in different environments eg:- TCP-MD5 [RFC2385] is used to secure a BGP session. The TCP option space limitation is imposed by the 4 bit Data Offset (DO) field which restricts the total length of the TCP header (including options) to 60 bytes ([RFC0793]). Since the required fields of the TCP header consumes 20 bytes, this leaves 40 bytes as the maximum amount of space for use by TCP options.

2.1. Motivation

The TCP option space of 40 bytes which was considered reasonable for the concurrent usage of some popular TCP options, is no longer able to cater to needs of recent growing demands. Over the past few years several new TCP options like TCP-UTO ([RFC5482]), TCP-AO ([RFC5925]), experimental TCP options for acknowledgment congestion control ([RFC5690]), have been proposed. MultiPath TCP ([I-D.ietf-mptcp-multiaddressed]) is another feature which calls for more TCP option space. Lastly middleboxes doing WAN optimization functionality can also benefit with the increased TCP option space, although it is not the intention of this memo to address such requirements. It is mentioned here simply for the sake of

completeness.

Since it is not easy to extend TCP option space, TCP option space hungry applications could move to a different protocol like SCTP ([RFC4960]), which for instance provides 255 chunks (each SCTP chunk is analogous to a TCP option) Needless to mention, this is not always possible and moving to a new protocol, at the least not only has all the issues that is exhibited by a new TCP option but even more (like general protocol upgrade and support, etc.,)

It is therefore expedient to extend the TCP option space to meet the near term requirements coming from the various TCP applications for active internet deployment and research.

3. Common issues with TCP option extension

Typically any proposal trying to extend the TCP option space needs to use some form of explicit signaling during the 3 way handshake of TCP to negotiate the usage of TCP extended options. This is typically accomplished by means of having a new TCP option or by using one of the reserved bits of the TCP header. In the reminder of the text we refer both of these methods simply as "TCP extended option(s)". The following notes apply to either of these methods.

Any TCP option extension proposals needs to be aware of the following design issues :-

End host backward compatibility - Any of the TCP option space extension proposals needs to satisfy the following end-host compatibility requirements :-

H1) Needs to have the capability to interoperate with stacks that do not have ability to negotiate the TCP extended option space.

H2) If the TCP extended options could not be negotiated, then it needs to fall back to normal mode of operation.

H3) It also needs to be noted that in some rare cases end-hosts not understanding a TCP option simply echoes back the TCP option in response. This is due to buggy implementation and hence this is not a requirement at all. Just mentioned for the sake of completeness.

Middle box interoperability - This is the toughest of the problems in getting towards a cleaner solution for extending TCP option space. It needs to be noted that middlebox interactions disrupt the end-to-end flow characteristics and any solution is susceptible to it's presence. TCP option space extension proposals need to be aware of the following issues :-

M1) Some middleboxes terminate TCP connections (Performance Enhancing Proxies) [RFC3135]

M2) Some middleboxes examine TCP payloads (Virus scanner, firewall), some modify the TCP payloads (NAT ALG)

M3) In general middleboxes keep state which includes TCP sequence numbers, some of them (proxies which do not terminate the TCP connection) would adjust the sequence and acknowledgement numbers on a per packet basis.

M4) Some middleboxes strip unknown TCP options.

M5) Some middleboxes resegment TCP data.

M6) Some middleboxes drop packets containing unknown TCP options.

Among the above M4, M5 and M6 are not common, M1, M2 & M3 exhibits varying degrees of commonality. Cases M2 and to some extent M3 proves to be the toughest candidate to work with for seamless TCP option extension.

Another point worthy of mention would be that, the outgoing path or the return path may change between connections or even during a connection, which can lead to traversal of different middleboxes. This simply means that the same middlebox behavior cannot be

guaranteed for all the packets of a given TCP connection.

4. TCP option space extension proposals

The following section summarizes some of the existing TCP option space proposals (section 4.1 to 4.4), touches upon some new ideas and variations of the existing proposals (section 4.5 to 4.7), and analyses each one of the proposal's strengths and weaknesses. It needs to be emphasized that the new ideas (section 4.5 to 4.7) are by no means a complete solution, but some general ideas which can be

used as building blocks or food for thought when trying to devise solution for the TCP option space exhaustion issue.

4.1. TCP LO and SLO

This ID ([I-D.eddy-tcp-loo]) talks about the defining a new TCP option that overrides the standard TCP DO field definition. This is accomplished by means of a TCP Long Option (LO) which gets negotiated during SYN. If both ends understand the LO option, then whenever the LO option is present, the DO field of the standard TCP header would be ignored for computing the TCP data offset. Instead the "Header Length" specified in the LO option would be used to derive the Data Offset. The LO option MUST be the first option present in the TCP segment. The draft actually mandates the LO option to be present in all segments, to circumvent odd bugs arising due to TCP segments with different interpretations of DO with and without the presence of the LO option.

The draft also defines a SYN Long Option (SLO). The purpose of this option is to retain backward compatibility with hosts that do not support the LO option. This is needed because a host which does not understand the LO option would mistakenly treat anything past the DO-specified boundary to be application data. The SLO option is used in the case where 40 bytes (max option space in standard TCP) is not enough to carry the desired TCP options to be sent on the SYN. It is only used by devices issuing an active open, since the passive open side can always use the LO option to send the long list of TCP options. The left-off options are reliably delivered in the subsequent data (and acknowledgment) segments.

Discussion :

This proposal addresses the host backward compatibility well (H1 & H2), since if the LO is not understood or stripped by a middlebox in between it would simply proceed with standard TCP semantics. However there is no provision for handling H3. This can be circumvented if needed, but it not probably worth the effort since such hosts are very uncommon in author's viewpoint. In the reminder of the proposals H3's compatibility is not going to be discussed.

Coming to the middlebox case, if the middlebox strips the TCP option

(M4), it is fine, since this option would not be negotiated. If the SYN gets dropped by a middlebox (M6), then retransmission request may be tried without the long option. For the case of PEP's (M1) if the connection gets terminated, then the middlebox which doesn't understand the LO option would reply with SYN+ACK without LO and it is fine since fallback to normal mode would happen. But this proposal (like any other proposal) may have implications with resegmenting (M5) feature of some middleboxes (which do not understand the LO option) since the TCP options gets blindly copied in all segmented segments by the middlebox. This would cause the end-host understanding the LO option to get confused, since it would interpret the duplicated segments as containing the TCP options.

The issue of option data getting confused as payload due the middlebox segmenting the TCP data can be circumvented by either having the total length of the TCP segment OR the TCP sequence number which points to the beginning of the data (instead of the proposed header length field which points to the Data Offset). It also needs to be noted that it is strongly advisable for any TCP options that could not fit into the standard TCP option space (40 bytes) begin after the 40 bytes boundary since any middlebox that cannot understand the LO option and does some TCP header fields sanity can get confused and drop the segment, if the new option is not fully contained in the TCP option space.

This proposal would have an issue with middleboxes that do not understand the extension and tries operate on the TCP payload (M2). For M3, it really depends on the middlebox, if the middlebox sees the next sequence number as lesser than the one it expects, it would simply treat that as a duplicate (good thing), however for some middleboxes if it tries to cache segments and perform retransmissions etc., it may be a problematic.

4.2. TCP DO field overload (Extended segments)

This proposal ([I-D.kohler-tcpm-extopt]) tries to solve the TCP option space issue by redefining the TCP DO field. Extended segments approach overloads the meaning of the standard TCP Data Offset field, keeping its original meaning for values of 5 and greater, but redefining it for values less than 5. This is seen as acceptable since values less than 5 are currently impossible, illegal, and unusable. Extended segments avoid the need for new options by changing the way that the existing standard header is parsed.

The granularity of option lengths that extended segments can support is limited to the number of unusable Data Offset values (5, 0 through 4). Currently, the extended segments proposal defines 4 fixed

lengths, and one "infinite" length that means the entire segment is options, with no application data. The fixed option lengths are 48, 64, 128, and 256 bytes.

Discussion :

If the required per-data-segment TCP options space for some extension or combination of extensions does not map to exactly one of the extended DO values, then padding bytes are required. If 129 bytes of options are required on a data segment, then a length of 256 must be used, and 127 bytes of useless padding are added.

As far as the end host compatibilty goes, if the end host doesn't understand the extended SYN (i.e. a SYN with DO < 5), the SYN would get dropped which is the typical behavior. Instead if some implementations who chose to send RST due to malformed segment (TCP header validation failure) is problematic since the end host would get confused whether the RST is due to its original intention (i.e. listener not present on the server or some other known thing like policy restrictions). The draft also mentions that extended SYN-ACKs may be sent in response to non-extended SYNs. This complicates the recovery procedure even more, if not understood, and goes against the way that all current negotiable TCP extensions operate (only used on SYN-ACK if advertised on SYN). Hence we can see that the fallback (cases H1 and H2) is not smooth when compared to the LO proposal.

As the middlebox goes, it suffers the same set of issues as the LO proposal. But there is no provision that can be made for the case of re-segmentation by middleboxes (M5). So, extended segments approach seems to be inferior compared to LO proposal in terms of robustness.

4.3. Increase (Double) TCP header size - TCPx2

This proposal([I-D.allman-tcpx2-hack]) was aimed to not just address the TCP option space exhaustion but to address other myriad of issues surrounding the TCP header fields. (like window size, reserved bits etc.,) TCPx2 requires a new IP protocol number. It defines a new TCP header which has all the original field sizes doubled. Hence the TCP DO field becomes 8 bits instead of the original 4 bits.

Discussion :

Ideally this proposal seems to be the best long term solution for TCP issues caused due the limits imposed by the various fields of TCP header, which includes the TCP option space issue. However, for short, medium or immediate term deployment it is not practical.

TCPx2 would face the same resistance (even worse since it is a new entrant) from middleboxes as experienced by some "newer" protocols

like SCTP etc., which hampered their successful adoption and deployment. End hosts TCP/IP stacks need to change to support the new IP protocol number, demux logic and new TCP header. The standard algorithms like TCP checksum, TCP MAC computation etc., needs to change, especially makes it difficult in cases where this functionality is assisted by hardware. The sockets API or TCP API's would have to change to accommodate the TCP header changes (src and dest ports) and also to offer backward compatibility to existing TCP. All the diagnostic tools surrounding etc., needs to change, the list only continues.

4.4. TCP LOIC

The crux of this proposal ([I-D.yourtchenko-tcp-loic]) is to quickly negotiate the LO (referred as LOIC in the document) option by sending it in SYN segment and also fallback to normal mode if the remote end or the middlebox drops the SYN. This is accomplished by sending 2 SYN's, one with the intent of negotiating the LO option and other one which contains the all the options that needs to be negotiated (includes the ones that cannot fit in the standard option space). To prevent the TCP option space mistakenly read as data, the proposal deliberately increments the checksum by 2 in the second SYN segment (which contains the TCP options). If the end host (or middlebox) doesn't understand the signalling (checksum + 2), it would drop the segment and the first SYN would get replied with SYN+ACK without LO. OTOH, if the stack understands the method, it would be able to process the both the SYNs (all the TCP options) and reply back with LO.

Discussion

This proposal cleverly avoids doing the option exchange dance after the SYN segment gets ACKnowledged like seen in LO proposal (section 4.1) by sending duplicate SYN's. This proposal addresses the host backward compatibility well (H1 & H2). However it has some issues. The overloading of checksum can have other implications esp., with some middeboxes that modify the checksum (NAT) which can rollover the 16 bit field and the checksum decrement would fail to work. Overloading checksum would not bode well with diagnostic tools

(sniffers etc.,) and also in cases where checksum gets offloaded to some hardware which needs to understand this method.

M1, M4, M6 are taken care of well by this proposal. It suffers the same fate as other proposals as far as M5 is concerned. It exhibits the same behavior as far as M2 and M3 goes, since the proposal sends TCP options as part of the standard TCP header.

One other note worthy of mentioning would be, this proposal is in

some sense similar to the concept of "extended segments". Extended segments approach tries to overload (use the illegal unused values) the DO field. One could argue that instead of mucking around with the checksum you could actually send the second SYN with "incorrect" (DO < 5) value. However if we throw in the middlebox into the equation, the likelihood of a packet getting dropped by the middlebox (firewall which does some TCP header validation) with DO field incorrectly specified is more when compared to packet with incorrect checksum. (This is because typically firewalls don't try to do compute intensive operations like TCP checksum validation)

4.5. Multiple segments with continuation

Both, extended segments and LOIC (section 4.2 and section 4.4) already mentions about sending multiple SYNs. This idea also uses the same approach with a key difference, i.e. send multiple SYNs with a "TCP continuation option". The TCP continuation option's purpose (which is a simple option-kind option) is to convey to the other end that "not all TCP options could be fit into to the current TCP segment's standard option space (40 bytes) and subsequent segments would convey the reminder of the option". For example lets us say we have 100 bytes of TCP option, this would require 3 segments to convey, the first 2 of which would have the TCP continuation option set. The rationale for this kind of thinking is twofold. Firstly, keeping in mind short term goals, TCP option space requirement isn't very large, hence a few segments should do the task at hand. Secondly, middleboxes (and end hosts) gets confused if you send something in the data portion, hence don't take chance of putting anything in the data portion of TCP.

The obvious drawback of this scheme is that if there a way too many TCP options to be conveyed, then it would take more segments to

convey all of them. For SYN segments, send multiple SYNs, for Data segments (during the middle of connection) try to send multiple data segments (behave as though nagle is turned off) and piggyback TCP options with TCP continuation. Well, this may sound outrageous but it should at least get rid of the middlebox issue at the cost of adding more processing and delay. Pick the devil.

Sending duplicate segments is generally ok and should not have adverse middlebox effects. Sending old segments (ones that are already ACKed is another school of thought. This is how keepalives work where the sequence number is set to TCB.SNDUNA - 1, which solicits a corrective ACK from the other end. Sending old data segment, simply gets dropped and one could use that to convey TCP options, but that may get clumsier since you really need TCP options to be inline with the current data. Just mentioned here for the sake of completeness.

4.6. TCP option cookies

This is yet another interesting thought (at least in author's opinion). The idea is to pack TCP options in a similar way TCP SYN cookies did. i.e., instead of doing the "Option-Kind" OR "option-kind-value", we could just have bit masks OR some encoding scheme defined under a TCP option template. We can call it as a "TCP template or TCP Master option". At least during the initial negotiation this would be helpful to scavenge TCP option space. The details of this need to be thought of and extensibility also needs to be addressed. Compressing TCP options is another school of thought along the similar lines, but the compression overhead needs space as well, hence needs to be carefully thought about.

4.7. Reuse/overload of other TCP fields

Some of the schemes already tried to overload some TCP fields like TCP DO (section 4.2) and TCP checksum overload (section 4.4). Another tempting field to overload is the TCP urgent pointer. It needs to be noted that the TCP urgent pointer is valid only when the URG bit is set. Also the TCP urgent pointer is used only by some legacy applications like Telnet and rlogin, both of which don't use any advanced TCP options nor require such a functionality. Also ([RFC6093]), suggests deprecating its usage for newer applications. It mentions that due to inconsistent interpretation of the urgent

pointer by end hosts and some middleboxes which clear the urgent flag and urgent pointer, the urgent mechanism should be deprecated. But, in authors opinion, the middleboxes that clear the urgent pointer only for some given ports like telnet and FTP. Given all these facts, it is not unreasonable to try to put this field to some good use. The urgent pointer could be used to redefine the TCP DO field in the same way the TCP DO redefinition scheme did but with a 16 bits of precision as opposed to just 3 (values < 5). The hope is that middleboxes (firewalls) would simply ignore the urgent pointer value since the URG bit is not set. We could also use the Urgent pointer for a fallback mechanism like the TCP checksum (LOIC) proposal did.

4.8. Miscellaneous

The previous sections have been trying to do a best possible effort in identifying some candidates for scavenging TCP option space, some ways of getting around the middleboxes etc., One school of thought would be, it is sometimes ok to convey TCP options across multiple segments (the TCP continuation option touched upon this aspect). for example let's say there are 5 SACK blocks, they can't be conveyed using a single TCP segment, this takes multiple TCP segments which is ok (the way it is working today). It may be possible to do this for other TCP options as well, which need not be conveyed on every packet

like the TCP security option. For e.g., TCP timestamps need to be conveyed on every segment, but care needs to be taken not to break PAWS etc.,

5. Conclusion

If we get this far in the document, the obvious question would be "is there a winner?" clearly not since any method has some issue or the other and most of them are vulnerable to the middlebox processing of packets (in particular cases M2 and M3 are tough to circumvent with). But, there are different types of middleboxes and it really needs to be seen what percentage of them exists in different environments. There was a recent interesting study [IMC2011], which focused on the issues of extending TCP which includes the analysis of various middlebox behaviors. This could be used as a reference point for any

new proposals on extending TCP which includes the TCP option space extension proposals.

This document listed a plethora of solutions (some of which may be half-baked) to address some common middlebox problems and graceful fallback to regular TCP methods. Now, no solution is a perfect solution and it is impossible to work around all the use cases imposed by middleboxes. Moving to a new protocol also is not a solution, hence we are left with no choice but to extend TCP that works for most of the use cases. Lastly, but not the least, the hope of this document is that it serves as a foundational document that touches upon the various aspects of extending TCP option space and acts as a motivational document to converge to a possible solution for the TCP option space exhaustion issue.

6. IANA Considerations

None.

7. Security Considerations

None at this time.

8. Acknowledgements

Thanks to Dan Wing and Andrew Yourtchenko for the initial review of the draft.

9. References

9.1. Normative References

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

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

9.2. Informative References

[I-D.allman-tcpx2-hack]
Allman, M., "TCPx2: Don't Fence Me In", draft-allman-tcpx2-hack-00 (work in progress), May 2006.

[I-D.eddy-tcp-loo]
Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[I-D.ietf-mptcp-multiaddressed]
Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-multiaddressed-07 (work in progress), March 2012.

[I-D.kohler-tcpm-extopt]
Kohler, E., "Extended Option Space for TCP", draft-kohler-tcpm-extopt-00 (work in progress), September 2004.

[I-D.yourtchenko-tcp-loic]

Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

[IMC2011] Honda et. al, M., "Internet Measurement conference 2011. http://conferences.sigcomm.org/imc/2011/docs/p181.pdf", November 2011.

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

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,

Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option", RFC 5482, March 2009.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.

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

[RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, January 2011.

Author's Address

Anantha Ramaiah
Cisco
170 Tasman Drive
San Jose, CA 95134
USA

Phone: +1 (408) 525-6486
Email: ananth@cisco.com