TCPM
Internet-Draft
Intended Status: Standards Track
File: draft-borman-tcp4way-00.txt
Expires: April 14, 2015

D. Borman
Quantum Corporation

October 14, 2014

TCP Four-Way Handshake

Abstract

One of the limitations of TCP is that it has limited space for TCP options, only 54 bytes. Many mechanisms have been proposed for for extending the TCP option space, but the biggest challenge has been to get additional option space in the initial SYN packet.

This memo presents a optional four-way TCP handshake to allow extended option space to be used in SYN packets in both directions.

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 April 14, 2015.

Table of Contents

1. Introduction

The TCP packet format has 54 bytes for adding TCP options. The most common method to extend TCP is to define new options, but the limited TCP option space can make that difficult as the number of potential options grow. Support for various TCP options is typically negotiated during the three-way handshake, in the packets that contain the SYN. If both sides send and receive a given option in a packet with the SYN bit set, then both sides know that the option is supported.

The majority of TCP sessions begin with three-way handshake, the exception to that is a simultaneous open.

The ideas presented in this memo were first hinted at in a message to the TCPM mailing list [Borman14].

2. Motivation For this Approach

The problem of expanding the TCP option space in the initial SYN packet has vexed designers for years. The main issue is maintaining compatibility with legacy TCP implementations, which don't understand the expanded TCP option space. When the initial SYN is sent, there is no knowledge as to whether or not the remote side can understand the extended option space. Various approaches have been considered:

1) Send dual SYNs, with and without the extended options, and arrange that the extended SYN will be considered invalid and dropped by legacy implementations. [Yourtchenko11] [Briscoe14]
2) Send an additional out of band packet along with the SYN to contain additional options. [Touch14]
3) Send additional options that didn't fit into the SYN in additional packets using a new TCP option. [Eddy08]
4) Send an initial SYN with extended options that a legacy server will fail, and then fall back to a new SYN without extended options. [Kohler04]

[Ramaiah12] contains additional analysis of proposed ways to expand the TCP option space.

Expanding the TCP option space in the initial SYN is a case of the more general issue: How can you change the fixed TCP header in the initial SYN packet and still maintain compatibility with legacy implementations? The TCP Window Scale option [RFC7323] redefined the Window field, but only in non-SYN packets. In the case of expanding the TCP option space, it involves redefining or overriding the Data Offset (DO) field.

The most straight forward method for dealing with modifying the initial SYN packet is to add an initial packet exchange so that the client can find out what the server supports, and then it knows, for example, if the server supports extended TCP option space. The problem with this approach is that it adds an additional RTT to connection startup, and most people are looking for ways to shorten, not lengthen, the initial connection setup, for example "TCP Fast Open" [TFO]. Though to be clear, the extra RTT is really not a concern about connection setup, but about when data can be first delivered to the application.

An alternative is to send the additional data in the initial SYN such that a legacy TCP will ignore it. This is most commonly done by sending the information in a TCP option, which legacy TCP would ignore. But the TCP option space is only 54 bytes, and by definition an expanded TCP option space won't fit in the legacy TCP option space. So, the additional data needs to be sent by some other

mechanism, e.g. in a second SYN or in an additional non-SYN packet. Challenges with this approach include the SYNs being routed to

different destination machines, the order of the packets being reversed, as well as a server needing to wait some amount of time to decide whether or not the additional packet will be arriving.

The goal of this proposal is to integrate discovery of server capabilities into the connection setup, while still allowing for data to be delivered in a timely manner.

3. TCP Four-Way Handshake

3.1 Overview

For a connection with ISS (Initial Send Sequence) values of ISSA from the client and ISSB from the server, the normal three-way TCP handshake is:

```
    Enter SYN-SENT
    SYN(seq=ISSA) ->
                                  Enter SYN-RECEIVED
                               <- SYN(seq=ISSB)/ACK(ISSA)
    Enter ESTABLISHED
    ACK(ISSB) ->
                                  Enter ESTABLISHED
```

A simultaneous open is:

```
    Enter SYN-SENT                Enter SYN-SENT
    SYN(seq=ISSA) ->           <- SYN(seq=ISSB)

    Enter SYN-RECEIVED            Enter SYN-RECEIVED
    SYN(seq=ISSA)/ACK(ISSB) ->  <- SYN(seq=ISSB)/ACK(ISSA)

    Enter ESTABLISHED             Enter ESTABLISHED
```

See [RFC793] page 68 and [RFC1122] page 86.

The normal scenario for the proposed four-way handshake is:

```
    Enter SYN-SENT
    SYN(seq=ISSA) ->
                                  Enter SYN-SENT
                               <- SYN(seq=ISSB)/ACK(ISSA)
    Enter SYN-RECEIVED
    SYN(seq=ISSA)/ACK(ISSB) ->
                                  Enter ESTABLISHED
                               <- ACK(ISSA)
    Enter ESTABLISHED
```

There are other options for the initial server response in the four-

way handshake. Those are discussed in Appendix A as well as the

reasons they weren't chosen.

3.2 Changes to the TCP state diagram

The changes can be described entirely as new new state transitions and some additional decisions:

```
    LISTEN -> rcv SYN,
        if (allow4way)
            passive4way=1, snd SYN,ACK -> SYN-SENT
        else
            passive4way=0, snd SND,ACK -> SYN-RCVD
    SYN-SENT -> rcv ACK
        if (passive4way == 1)
            -> ESTABLISHED
        else
            normal error processing

    CLOSED -> active OPEN, create TCB, snd SYN,
                           active4way=1 -> SYN-SENT

    SYN-SENT -> rcv SYN,ACK
        if (active4way == 1 && (continue4way))
            snd SYN,ACK -> SYN-RCVD
        else
            snd ACK -> ESTABLISHED
```

The "allow4way" and "continue4way" decisions are based on the contents of the inbound packet.

Instead of overloading the SYN-SENT state and burying the decisions in the existing LISTEN and SYN-SENT states, the state diagram could be expanded with one new state, SYN-ACK-SENT, and two transitional states, ALLOW-4WAY and CONTINUE-4WAY. These *-4WAY states are transitional because once entered, an immediate decision is made and then they are immediately exited to a new state.

The LISTEN -> SYN-RCVD transition is replaced by:

```
    LISTEN -> rcv SYN -> ALLOW-4WAY

    ALLOW-4WAY(YES) -> snd SYN,ACK -> SYN-ACK-SENT
    ALLOW-4WAY(NO) -> snd SYN,ACK -> SYN-RCVD

    SYN-ACK-SENT -> rcv SYN,ACK, snd ACK -> ESTABLISHED
    SYN-ACK-SENT -> rcv ACK of SYN, x -> ESTABLISHED
```

and the SYN-SENT -> ESTABLISHED transition is replace by:

```
SYN-SENT -> rcv SYN,ACK -> CONTINUE-4WAY

CONTINUE-4WAY(YES) -> snd SYN,ACK -> SYN-RCVD
CONTINUE-4WAY(NO) -> snd ACK -> ESTABLISHED
```

3.3 Three-Way or Four-Way Handshake?

There are two new decision points for for handling a four-way handshake. First, when a connection in LISTEN state receives a SYN packet, it has to decide based on the contents of that packet whether or not the remote side understands the four-way handshake. This is accomplished through the allocation of one of the unused bits in the TCP header, the 4WAY bit.

> Note: Other ways to convey support for the four-way handshake were considered, these are discussed in Appendix B.

The client sets the 4WAY bit in the initial SYN. If the server receives a 4WAY bit in the initial SYN, then it will set the 4WAY bit in the SYN/ACK. If the client recieves a SYN/ACK without the 4WAY bit set, it proceeds with the normal three-way handshake. If it receives a SYN/ACK with the 4WAY bit set, then based on the options in the SYN/ACK it can chose to either proceed with the normal three-way handshake, or to continue with the four-way handshake.

If a packet is received with the 4WAY bit set, but not the SYN bit, the 4WAY bit is ignored. When sending a packet without the SYN bit set, the 4WAY bit must not be set.

[RFC3168] notes TCP interoperability issues with the CWR and ECE bits, but the 4WAY bit does not have the same issues.

3.3.1 Non Four-Way Client Sets 4WAY bit

In this case, the server might enter SYN-ACK-SENT state. It will respond with a SYN-ACK. Because this looks like the same ACK generated in SYN-RCVD state, it will look to the client like a normal SYN/ACK packet, other than the 4WAY bit, and it will respond with a normal ACK, and the connection will complete with the normal three-way handshake.

3.3.2 Non Four-Way Server Sets 4WAY bit

If the client decides to not continue a four-way handshake, then it will respond with an ACK and complete the normal three-way handshake. If the client decides that it does want to continue with a four-way exchange, it'll send a SYN/ACK. When the server receives the packet,

the normal TCP processing will strip off the SYN, and continue

processing as a normal three-way handshake.

3.4 RTT Costs of the Four-Way Handshake

When compared to the three-way handshake, the four-way handshake adds an additional 0.5 RTT before both sides enter ESTABLISHED state. But the more important question is how does the four-way handshake affect the delivery of initial data to the application? This is best answered by looking at some specific cases, comparing the three-way handshake with the four-way handshake.

Data can be sent on a SYN packet, but it cannot be delivered to the application until entering ESTABLISHED state.

Three-way handshake with data sent once in ESTABLISHED:

```
    0.0) SYN ->
    0.5)            <- SYN/ACK
    1.0) Client enters ESTABLISHED
         ACK w/client data ->
    1.5)            Server enters ESTABLISHED, delivers client data.
                    <- Server data
    2.0) Client delivers server data
```

Four-way handshake with data sent once in ESTABLISHED:

```
    0.0) SYN ->
    0.5)                <- SYN
    1.0) SYN/ACK->
    1.5)                Server enters ESTABLISHED state
                        ACK w/Server data
    2.0) Client enters ESTABLISHED state
         Client delivers server data
         ACK w/client data ->
    2.5)                Server delivers client data
```

So in both cases, the server data is delivered at the client after 2 RTTs, and for the four-way the client's data is delivered to the server at 2.5 RTT instead of 1.5 RTT, so 1 RTT later.

Now, let's look at both cases with data in the SYN/ACK:

Three-way handshake:

```
    0.0) SYN ->
    0.5)                <- SYN/ACK w/server-data
    1.0) Client enters ESTABLISHED
         Client delivers server-data
         ACK w/client data ->
```

1.5) Server enters ESTABLISHED

```
                        Server delivers client-data.
```

Four-way handshake:

```
    0.0) SYN ->
    0.5)                <- SYN
    1.0) SYN/ACK with client-data ->
    1.5)                Server enters ESTABLISHED
                        Server delivers client-data
                        <- ACK w/server data
    2.0) Client enters ESTABLISHED
         Client delivers server-data.
```

You get the same differences, but reversed. The client's data is delivered after 1.5 RTTs in both cases, and the servers data is delivered 1 RTT later, at 2.0 RTT instead of 1.0 RTT.

If you put the data with the bare SYN, the initial data doesn't get delivered any sooner, because you still have to wait for the ACK of the SYN to deliver the data.

4. Negotiating Non-directional vs. Directional TCP Options

TCP options that are negotiated in the initial SYN exchange can be classified as either non-directional or directional. An example of a non-directional option is the TCP Window Scale option. Negotiating a non-directional TCP option falls naturally into the Four-Way handshake, but allows for more options to be negotiated than will fit into the initial SYN packet when using expanded TCP option space. In order to allow this, the SYN/ACK from the server, with the TCP Extended Data option (EDO) [EDO], can contain initial negotiation for TCP options that weren't received in the initial SYN, which the client can then acknowledge in its SYN/ACK, using the EDO option. Because the options are non-directional, it doesn't matter which side presents it first.

Directional options do not fall as cleanly into the extended four-way handshake. A directional option is one which is originated in the initial SYN, and the servers response in the SYN/ACK is determined in direct response to the inbound option. For example, assume an option FOO that has 100 variants, where servers typically have support for all 100 variants, but clients usually only a small number. The client sends option FOO with a short list of variants that it supports, and then the server chooses which one of those to use, and responds with that variant. If instead the server initiates the the option in the SYN/ACK, it'd have to include all 100 variants and let the client choose from that list. In the future, new TCP options would need to be designed to work in the context of the four-way

handshake. For existing directional options, it would not be unreasonable to require that they be included in the initial SYN, and

other non-directional options would be deferred and negotiated in the SYN/ACK exchange.

5. TCP Connection State Diagram

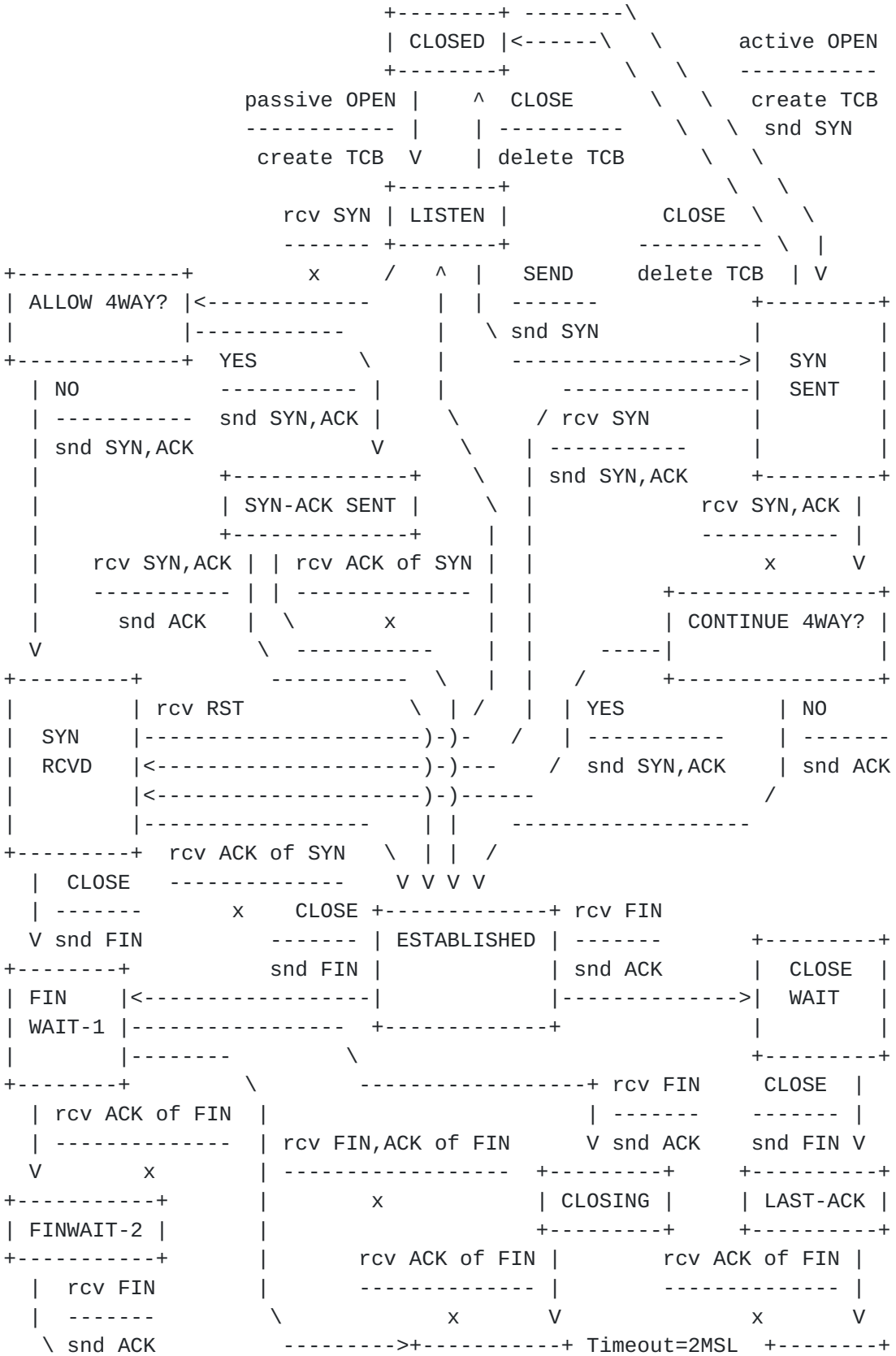
The following diagram is modified from the diagram in RFC 793 [RFC793]. In addition to adding the "ALLOW 4WAY?", "CONTINUE 4WAY?" and "SYN-ACK SENT" states, it also includes the three changes listed in RFC 1122 [RFC1122]:

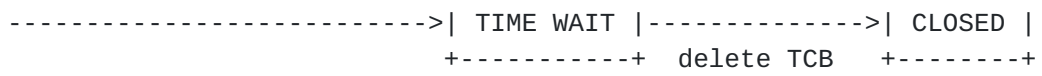
"(a) The arrow from SYN-SENT to SYN-RCVD should be labeled with "snd SYN,ACK", to agree with the text on page 68 and with Figure 8.

(b) There could be an arrow from SYN-RCVD state to LISTEN state, conditioned on receiving a RST after a passive open (see text page 70).

(c) It is possible to go directly from FIN-WAIT-1 to the TIME-WAIT state (see page 75 of the spec)."

TCP Connection State Diagram




----------->| TIME WAIT |----------->| CLOSED |
+-----------+ delete TCB +--------+

6. IANA Considerations

TBD

7. Security Considerations

TBD

8. References

8.1 Normative References

[RFC793] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

8.2 Informative References

[Borman14] Borman, D., "Re: [tcpm] New Version Notification for draft-touch-tcpm-tcp-edo-01.txt", message to the TCPM mailing list, 22 May 2014, <http://www.ietf.org/mail-archive/web/tcpm/current/msg08804.html>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

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

[TFO] Cheng, Y., Jhu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", Work in Progress, draft-ietf-tcpm-fastopen-10.txt, September 2014.

[EDO] Joe Touch, J., and W. Eddy, "TCP Extended Data Offset Option", Work in Progress, draft-ietf-tcpm-tcp-edo-01.txt, October 2014.

[Kohler04] Kohler, E, "Extended Option Space for TCP" Work in Progress, draft-kohler-tcpm-extopt-00.txt, September 2004.

[Touch14] Touch, J., Briscoe, B., and T. Faber, "TCP SYN Extended Option Space in the Payload of a Supplementary Segment", Work in Progress, draft-touch-tcpm-tcp-syn-ext-opt-01.txt, September 2014.

[Eddy08] Eddy, W., and A. Langley, "Extending the Space Available for TCP Options", Work in Progress, draft-eddy-tcp-loo-04, July 2008.

[Yourtchenko11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", Work in Progress, draft-yourtchenko-tcp-loic-00.txt, April 2011.

[Ramaiah12] Ramaiah, A., "TCP option space extension", Work in Progress, draft-ananth-tcpm-tcpoptext-00.txt, March 2012.

[Briscoe14] Briscoe, B., "Extended TCP Option Space in the Payload of an Alternative SYN", Work in Progress, draft-briscoe-tcpm-syn-op-sis-02, September 2014.

Appendix A. First Response of the Four-Way Handshake

For a connection with ISS values of ISSA from the client and ISSB from the server, three different options for the first server response were considered:

(1) SYN(seq=ISSB)
(2) SYN(seq=ISSB)/ACK(seq=ISSA-1)
(3) SYN(Seq=ISSB)/ACK(seq=ISSA)

SYN(seq=ISSB)

The original idea for the four-way handshake was to have the server do a simple turn-around of the TCP three-way handshake, by

responding to the initial SYN with another bare SYN. Because it had already received a SYN and knows that the client supports the four-way handshake, it could respond with a plain SYN, making use of header modifying options that the client had indicated it supported. This is similar to a a simultaneous open, except the server is able to transition from SYN-SENT to ESTABLISHED instead of going through SYN-RECEIVED state.

```
    Enter SYN-SENT
    SYN(seq=ISSA) ->
                                      Enter SYN-SENT
                                   <- SYN(seq=ISSB)
    Enter SYN-RECEIVED
    SYN(seq=ISSA)/ACK(ISSB) ->
                                      Enter ESTABLISHED
                                   <- ACK(ISSA)
    Enter ESTABLISHED
```

The problems with this approach are that it forces the full four-way handshake, and a middle-box in the path might block the returning bare SYN.

SYN(seq=ISSB)/ACK(seq=ISSA-1)

This response also turns the three-way handshake into something that looks a lot like a simultaneous open, since the ACK does not acknowledge the SYN. The disadvantage is that it also forces a full four-way handshake, since it does not acknowledge the initial SYN. However, this should work better for getting through a middle-box since it is not a bare SYN. But if the middle-box is digging into the TCP packet and tries to verify the ACK field, it might still block this packet since it is not the expected ACK field of the normal three-way handshake.

SYN(seq=ISSB)/ACK(seq=ISSA)

This response looks like the normal three-way handshake response, which gives the client the ability to choose whether to complete the three-way handshake by sending an ACK(ISSB), or continue the four-way handshake by responding with SYN(seq=ISSA)/ACK(ISSB). The advantage of this option is that it doesn't always force the four-way handshake, and to a middle-box the packets look like the normal TCP packets that it expects to see.

The third option offers the least possibility that middle-boxes will block the packets, and also leaves the flexibility for deciding on a three-way or four-way handshake up to the client. Because it is to the client's benefit to have a four-way handshake, it should be the

one to decide whether or not the four-way handshake is needed for a

particular handshake.

Appendix B. Communicating Four-Way Handshake Support

Besides allocating a 4WAY bit in the TCP header, two other options were considered for communicating support for the four-way handshake:

Create a new 4WAY TCP option

This does not have the interoperability issues that the 4WAY TCP bit has, because it is assumed that connections will not send unknown TCP options. The disadvantage of this is that it requires two more bytes out of the TCP option space.

Implied support by other TCP options

The primary motivation for the four-way handshake is to give the client a second chance to send TCP options in a SYN. This is intended for use with the new TCP EDO option, and the presence of the EDO option could imply support for the four-way handshake. This allows the client to send additional TCP options using the TCP EDO option in a SYN/ACK packet.

Acknowledgments

TBD

Contributors

TBD

Author's Address

David Borman
Quantum Corporation
1155 Centre Pointe Drive, Suite 1
Mendota Heights, MN 55120

Phone: (651) 688-4394
Email: david.borman@quantum.com