TCPM
Internet Draft
Intended status: Informational
Expires: April 2022

J. Touch
Independent consultant
J. Kuusisaari
Infinera

October 11, 2021

TCP-AO Test Vectors

draft-ietf-tcpm-ao-test-vectors-02.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 12, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This document provides test vectors to validate implementations of the two mandatory authentication algorithms specified for the TCP Authentication Option over both IPv4 and IPv6. This includes validation of the key derivation function (KDF) based on a set of test connection parameters as well as validation of the message authentication code (MAC). Vectors are provided for both currently required pairs of KDF and MAC algorithms: one based on SHA-1 and the other on AES-128. The vectors also validate both whole TCP segments as well as segments whose options are excluded for middlebox traversal.

Table of Contents

1. Introduction....3
2. Conventions used in this document....4
3. Input Test Vectors....4
   3.1. TCP Connection Parameters....4
      3.1.1. TCP-AO parameters....4
      3.1.2. Active (client) side parameters....4
      3.1.3. Passive (server) side parameters....5
      3.1.4. Other IP fields and options....5
      3.1.5. Other TCP fields and options....5
4. IPv4 SHA-1 Output Test Vectors....5
   4.1. SHA-1 MAC (default - covers TCP options)....6
      4.1.1. Send (client) SYN (covers options)....6
      4.1.2. Receive (server) SYN-ACK (covers options)....6
      4.1.3. Send (client) non-SYN (covers options)....7
      4.1.4. Receive (server) non-SYN (covers options)....7
   4.2. SHA-1 MAC (omits TCP options)....8
      4.2.1. Send (client) SYN (omits options)....8
      4.2.2. Receive (server) SYN-ACK (omits options)....8
      4.2.3. Send (client) non-SYN (omits options)....9
      4.2.4. Receive (server) non-SYN (omits options)....9
5. IPv4 AES-128 Output Test Vectors....10
   5.1. AES MAC (default - covers TCP options)....10
      5.1.1. Send (client) SYN (covers options)....10
      5.1.2. Receive (server) SYN-ACK (covers options)....11
      5.1.3. Send (client) non-SYN (covers options)....11
      5.1.4. Receive (server) non-SYN (covers options)....12
   5.2. AES MAC (omits TCP options)....12
      5.2.1. Send (client) SYN (omits options)....12

5.2.2. Receive (server) SYN-ACK (omits options).....13
5.2.3. Send (client) non-SYN (omits options).....13
5.2.4. Receive (server) non-SYN (omits options).....14
6. IPv6 SHA-1 Output Test Vectors.....14
6.1. SHA-1 MAC (default - covers TCP options).....15
6.1.1. Send (client) SYN (covers options).....15
6.1.2. Receive (server) SYN-ACK (covers options).....15
6.1.3. Send (client) non-SYN (covers options).....16
6.1.4. Receive (server) non-SYN (covers options).....16
6.2. SHA-1 MAC (omits TCP options).....17
6.2.1. Send (client) SYN (omits options).....17
6.2.2. Receive (server) SYN-ACK (omits options).....18
6.2.3. Send (client) non-SYN (omits options).....18
6.2.4. Receive (server) non-SYN (omits options).....19
7. IPv6 AES-128 Output Test Vectors.....19
7.1. AES MAC (default - covers TCP options).....19
7.1.1. Send (client) SYN (covers options).....19
7.1.2. Receive (server) SYN-ACK (covers options).....20
7.1.3. Send (client) non-SYN (covers options).....20
7.1.4. Receive (server) non-SYN (covers options).....21
7.2. AES MAC (omits TCP options).....22
7.2.1. Send (client) SYN (omits options).....22
7.2.2. Receive (server) SYN-ACK (omits options).....22
7.2.3. Send (client) non-SYN (omits options).....23
7.2.4. Receive (server) non-SYN (omits options).....23
8. Observed Implementation Errors.....24
8.1. Algorithm issues.....24
8.2. Algorithm parameters.....24
8.3. String handling issues.....24
8.4. Header coverage issues.....25
9. Security Considerations.....25
10. IANA Considerations.....25
11. References.....25
11.1. Normative References.....25
11.2. Informative References.....26
12. Acknowledgments.....26

1. Introduction

This document provides test vectors to validate the correct implementation of the TCP Authentication Option (TCP-AO) [RFC5925] and its mandatory cryptographic algorithms defined in [RFC5926]. It includes the specification of all endpoint parameters to generate the variety of TCP segments covered by different keys and MAC coverage, i.e., both the default case and the variant where TCP options are ignored for middlebox traversal. It also includes both default key derivation functions (KDFs) and MAC generation

algorithms [RFC5926] and lists common pitfalls of implementing the algorithms correctly.

The experimental extension to support NAT traversal is not included in the provided test vectors [RFC6978].

This document provides test vectors multiple implementations that have been validated against each other for interoperability.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Input Test Vectors

3.1. TCP Connection Parameters

The following parameters are used throughout this suite of test vectors. The terms 'active' and 'passive' are used as defined for TCP [RFC793].

3.1.1. TCP-AO parameters

The following values are used for all exchanges. This suite does not test key switchover. The KeyIDs are as indicated for TCP-AO [RFC5925]. The Master Key is used to derive the traffic keys [RFC5926].

Active (client) side KeyID: 61 decimal (0x3d hexadecimal)

Passive (server) side KeyID: 84 decimal (0x54 hexadecimal)

Master_key: "testvector" (length = 10 bytes)

3.1.2. Active (client) side parameters

The following endpoint parameters are used on the active side of the TCP connection, i.e., the side that initiates the TCP SYN.

For IPv4: 10.11.12.13 (dotted decimal)

For IPv6: fd00::1 (IPv6 hexadecmal)

TCP port: (varies)

3.1.3. Passive (server) side parameters

The following endpoint parameters are used for the passive side of the TCP connection, i.e., the side that responds with a TCP SYN-ACK.

For IPv4: 172.27.28.29 (dotted decimal)

For IPv6: fd00::2 (IPv6 hexadecimal

TCP port = 179 decimal (BGP)

3.1.4. Other IP fields and options

No IP options are used in these test vectors.

All IPv4 packets use the following other parameters [RFC791]: DSCP = 111000 binary (CS7) as is typical for BGP, ECN = 00 binary, set DF, and clear MF.

IPv4 uses a TTL of 255 decimal; IPv6 uses a hop limit of 64 decimal.

All IPv6 packets use the following other parameters [RFC8200]: traffic class = 0xe0 hexidecimal (DSCP = 111000 binary CS7, as is typical for BGP, with ECN = 00 binary) and no EHs.

3.1.5. Other TCP fields and options

The SYN and SYN-ACK segments include MSS [RFC793], NOP, WindowScale [RFC7323], SACK Permitted [RFC2018], TimeStamp [RFC7323], and TCP-AO [RFC5925], in that order.

All other example segments include NOP, NOP, TimeStamp, and TCP-AO, in that order.

All segment URG pointers are zero [RFC793]. All segments with data set the PSH flag [RFC793].

4. IPv4 SHA-1 Output Test Vectors

SHA-1 is computed as specified for TCP-AO [RFC5926].

In the following sections, all values are indicated as 2-digit hexadecimal values with spacing per line representing the contents of 16 consecutive bytes, as is typical for data dumps. The IP/TCP data indicates the entire IP packet, including the TCP segment and

its options (whether covered by TCP-AO or not, as indicated), including TCP-AO.

4.1. SHA-1 MAC (default - covers TCP options)

4.1.1. Send (client) SYN (covers options)

Send_SYN_traffic_key:

```
6d 63 ef 1b 02 fe 15 09 d4 b1 40 27 07 fd 7b 04
16 ab b7 4f
```

IPv4/TCP:

```
45 e0 00 4c dd 0f 40 00 ff 06 bf 6b 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5a 00 00 00 00
e0 02 ff ff ca c4 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 15 5a b7 00 00 00 00 1d 10 3d 54
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

MAC:

```
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

4.1.2. Receive (server) SYN-ACK (covers options)

Receive_SYN_traffic_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 4c 65 06 40 00 ff 06 37 75 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 61 fb fb ab 5b
e0 12 ff ff 37 76 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 84 a5 0b eb 00 15 5a b7 1d 10 54 3d
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

MAC:

```
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

4.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
d2 e5 9c 65 ff c7 b1 a3 93 47 65 64 63 b7 0e dc
24 a1 3d 71
```

IPv4/TCP:

```
45 e0 00 87 36 a1 40 00 ff 06 65 9f 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5b 11 c1 42 62
c0 18 01 04 a1 62 00 00 01 01 08 0a 00 15 5a c1
84 a5 0b eb 1d 10 3d 54 70 64 cf 99 8c c6 c3 15
c2 c2 e2 bf ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
70 64 cf 99 8c c6 c3 15 c2 c2 e2 bf
```

4.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 87 1f a9 40 00 ff 06 7c 97 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 62 fb fb ab 9e
c0 18 01 00 40 0c 00 00 01 01 08 0a 84 a5 0b f5
00 15 5a c1 1d 10 54 3d a6 3f 0e cb bb 2e 63 5c
95 4d ea c7 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
a6 3f 0e cb bb 2e 63 5c 95 4d ea c7
```

4.2. SHA-1 MAC (omits TCP options)

4.2.1. Send (client) SYN (omits options)

Send_SYN_traffic_key:

```
30 ea a1 56 0c f0 be 57 da b5 c0 45 22 9f b1 0a
42 3c d7 ea
```

IPv4/TCP:

```
45 e0 00 4c 53 99 40 00 ff 06 48 e2 0a 0b 0c 0d
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ee 00 00 00 00
e0 02 ff ff 54 1f 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 02 4c ce 00 00 00 00 1d 10 3d 54
80 af 3c fe b8 53 68 93 7b 8f 9e c2
```

MAC:

```
80 af 3c fe b8 53 68 93 7b 8f 9e c2
```

4.2.2. Receive (server) SYN-ACK (omits options)

Receive_SYN_traffic_key:

```
b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41a
01 a8 30 7f
```

IPv4/TCP:

```
45 e0 00 4c 32 84 40 00 ff 06 69 f7 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e1 cb 0e fb ef
e0 12 ff ff 38 8e 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 57 67 72 f3 00 02 4c ce 1d 10 54 3d
09 30 6f 9a ce a6 3a 8c 68 cb 9a 70
```

MAC:

```
09 30 6f 9a ce a6 3a 8c 68 cb 9a 70
```

4.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
f3 db 17 93 d7 91 0e cd 80 6c 34 f1 55 ea 1f 00
34 59 53 e3
```

IPv4/TCP:

```
45 e0 00 87 a8 f5 40 00 ff 06 f3 4a 0a 0b 0c 0d
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ef ac d5 b5 e2
c0 18 01 04 6c 45 00 00 01 01 08 0a 00 02 4c ce
57 67 72 f3 1d 10 3d 54 71 06 08 cc 69 6c 03 a2
71 c9 3a a5 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
71 06 08 cc 69 6c 03 a2 71 c9 3a a5
```

4.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41
01 a8 30 7f
```

IPv4/TCP:

```
45 e0 00 87 54 37 40 00 ff 06 48 09 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e2 cb 0e fc 32
c0 18 01 00 46 b6 00 00 01 01 08 0a 57 67 72 f3
00 02 4c ce 1d 10 54 3d 97 76 6e 48 ac 26 2d e9
ae 61 b4 f9 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
97 76 6e 48 ac 26 2d e9 ae 61 b4 f9
```

5. IPv4 AES-128 Output Test Vectors

AES-128 is computed as required by TCP-AO [RFC5926]

In the following sections, all values are indicated as 2-digit hexadecimal values with spacing per line representing the contents of 16 consecutive bytes, as is typical for data dumps. The IP/TCP data indicates the entire IP packet, including the TCP segment and its options (whether covered by TCP-AO or not, as indicated), including TCP-AO.

5.1. AES MAC (default - covers TCP options)

5.1.1. Send (client) SYN (covers options)

Send_SYN_traffic_key:

```
f5 b8 b3 d5 f3 4f db b6 eb 8d 4a b9 66 0e 60 e3
```

IP/TCP:

```
45 e0 00 4c 7b 9f 40 00 ff 06 20 dc 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d df 00 00 00 00
e0 02 ff ff 5a 0f 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 7e d0 00 00 00 00 1d 10 3d 54
e4 77 e9 9c 80 40 76 54 98 e5 50 91
```

MAC:

```
e4 77 e9 9c 80 40 76 54 98 e5 50 91
```

5.1.2. Receive (server) SYN-ACK (covers options)

Receive_SYN_traffic_key:

```
4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1
```

IPv4/TCP:

```
45 e0 00 4c 4b ad 40 00 ff 06 50 ce ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d e9 78 7a 1d e0
e0 12 ff ff f3 f2 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 93 f4 e9 e8 00 01 7e d0 1d 10 54 3d
d6 ad a7 bc 4c dd 53 6d 17 69 db 5f
```

MAC:

```
d6 ad a7 bc 4c dd 53 6d 17 69 db 5f
```

5.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
8c 8a e0 e8 37 1e c5 cb b9 7e a7 9d 90 41 83 91
```

IPv4/TCP:

```
45 e0 00 87 fb 4f 40 00 ff 06 a0 f0 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d e0 fa dd 6d ea
c0 18 01 04 95 05 00 00 01 01 08 0a 00 01 7e d0
93 f4 e9 e8 1d 10 3d 54 77 41 27 42 fa 4d c4 33
ef f0 97 3e ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
77 41 27 42 fa 4d c4 33 ef f0 97 3e
```

5.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1
```

IPv4/TCP:

```
45 e0 00 87 b9 14 40 00 ff 06 e3 2b ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d ea 78 7a 1e 23
c0 18 01 00 e7 db 00 00 01 01 08 0a 93 f4 e9 e8
00 01 7e d0 1d 10 54 3d f6 d9 65 a7 83 82 a7 48
45 f7 2d ac ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
f6 d9 65 a7 83 82 a7 48 45 f7 2d ac
```

5.2. AES MAC (omits TCP options)

5.2.1. Send (client) SYN (omits options)

Send_SYN_traffic_key:

```
2c db ae 13 92 c4 94 49 fa 92 c4 50 97 35 d5 0e
```

IPv4/TCP:

```
45 e0 00 4c f2 2e 40 00 ff 06 aa 4c 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 71 00 00 00 00
e0 02 ff ff 70 bf 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 85 e1 00 00 00 00 1d 10 3d 54
c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49
```

MAC:

```
c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49
```

5.2.2. Receive (server) SYN-ACK (omits options)

Receive_SYN_traffic_key:

```
3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a
```

IPv4/TCP:

```
45 e0 00 4c 6c c0 40 00 ff 06 2f bb ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 6f 38 9b ed 72
e0 12 ff ff e4 45 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a ce 45 98 38 00 01 85 e1 1d 10 54 3d
3a 6a bb 20 7e 49 b1 be 71 36 db 90
```

MAC:

```
3a 6a bb 20 7e 49 b1 be 71 36 db 90
```

5.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
03 5b c4 00 a3 41 ff e5 95 f5 9f 58 00 50 06 ca
```

IPv4/TCP:

```
45 e0 00 87 ee 91 40 00 ff 06 ad ae 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 72 d3 84 4a 70
c0 18 01 04 88 51 00 00 01 01 08 0a 00 01 85 e1
ce 45 98 38 1d 10 3d 54 75 85 e9 e9 d5 c3 ec 85
7b 96 f8 37 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
75 85 e9 e9 d5 c3 ec 85 7b 96 f8 37
```

5.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a
```

IPv4/TCP:

```
45 e0 00 87 6a 21 40 00 ff 06 32 1f ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 70 38 9b ed 72
c0 18 01 00 04 49 00 00 01 01 08 0a ce 45 98 38
00 01 85 e1 1d 10 54 3d 5c 04 0f d9 23 33 04 76
5c 09 82 f4 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
5c 04 0f d9 23 33 04 76 5c 09 82 f4
```

6. IPv6 SHA-1 Output Test Vectors

SHA-1 is computed as specified for TCP-AO [RFC5926].

6.1. SHA-1 MAC (default - covers TCP options)

6.1.1. Send (client) SYN (covers options)

Send_SYN_traffic_key:

```
62 5e c0 9d 57 58 36 ed c9 b6 42 84 18 bb f0 69
89 a3 61 bb
```

IPv6/TCP:

```
6e 08 91 dc 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 3f
00 00 00 00 e0 02 ff ff 47 21 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 41 d0 87 00 00 00 00
1d 10 3d 54 90 33 ec 3d 73 34 b6 4c 5e dd 03 9f
```

MAC:

```
90 33 ec 3d 73 34 b6 4c 5e dd 03 9f
```

6.1.2. Receive (server) SYN-ACK (covers options)

Receive_SYN_traffic_key:

```
e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7
71 eb cb 4a
```

IPv6/TCP:

```
6e 01 00 9e 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4b
17 6a 83 40 e0 12 ff ff bf ec 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a bd 33 12 9b 00 41 d0 87
1d 10 54 3d f1 cb a3 46 c3 52 61 63 f7 1f 1f 55
```

MAC:

```
f1 cb a3 46 c3 52 61 63 f7 1f 1f 55
```

6.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
1e d8 29 75 f4 ea 44 4c 61 58 0c 5b d9 0d bd 61
bb c9 1b 7e
```

IPv6/TCP:

```
6e 08 91 dc 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 40
3f 51 99 4c c0 18 01 00 32 9c 00 00 01 01 08 0a
00 41 d0 91 bd 33 12 9b 1d 10 3d 54 bf 08 05 fe
b4 ac 7b 16 3d 6f cd f2 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
bf 08 05 fe b4 ac 7b 16 3d 6f cd f2
```

6.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7
71 eb cb 4a
```

IPv6/TCP:

```
6e 01 00 9e 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4c
17 6a 83 83 c0 18 01 00 ee 6e 00 00 01 01 08 0a
bd 33 12 a5 00 41 d0 91 1d 10 54 3d 6c 48 12 5c
11 33 5b ab 9a 07 a7 97 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
6c 48 12 5c 11 33 5b ab 9a 07 a7 97
```

6.2. SHA-1 MAC (omits TCP options)

6.2.1. Send (client) SYN (omits options)

Send_SYN_traffic_key:

```
31 a3 fa f6 9e ff ae 52 93 1b 7f 84 54 67 31 5c
27 0a 4e dc
```

IPv6/TCP:

```
6e 07 8f cd 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 69
00 00 00 00 e0 02 ff ff a4 1a 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 9d b9 5b 00 00 00 00
1d 10 3d 54 88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

MAC:

```
88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

6.2.2. Receive (server) SYN-ACK (omits options)

Receive_SYN_traffic_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

IPv6/TCP:

```
6e 0a 7e 1f 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4d
02 0c 1e 6a e0 12 ff ff 77 4d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 5e c9 9b 70 00 9d b9 5b
1d 10 54 3d 3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

MAC:

```
3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

6.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
b3 4e ed 6a 93 96 a6 69 f1 c4 f4 f5 76 18 f3 65
6f 52 c7 ab
```

IPv6/TCP:

```
6e 07 8f cd 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 6a
eb a3 73 4e c0 18 01 00 83 e6 00 00 01 01 08 0a
00 9d b9 65 5e c9 9b 70 1d 10 3d 54 48 bd 09 3b
19 24 e0 01 19 2f 5b f0 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
48 bd 09 3b 19 24 e0 01 19 2f 5b f0
```

6.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

IPv6/TCP:

```
6e 0a 7e 1f 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4e
02 0c 1e ad c0 18 01 00 71 6a 00 00 01 01 08 0a
5e c9 9b 7a 00 9d b9 65 1d 10 54 3d 55 9a 81 94
45 b4 fd e9 8d 9e 13 17 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
55 9a 81 94 45 b4 fd e9 8d 9e 13 17
```

7. IPv6 AES-128 Output Test Vectors

AES-128 is computed as required by TCP-AO [RFC5926].

7.1. AES MAC (default - covers TCP options)

7.1.1. Send (client) SYN (covers options)

Send_SYN_traffic_key:

```
fa 5a 21 08 88 2d 39 d0 c7 19 29 17 5a b1 b7 b8
```

IP/TCP:

```
6e 04 a7 06 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ec
00 00 00 00 e0 02 ff ff de 5d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 13 e4 ab 99 00 00 00 00
1d 10 3d 54 59 b5 88 10 74 81 ac 6d c3 92 70 40
```

MAC:

```
59 b5 88 10 74 81 ac 6d c3 92 70 40
```

7.1.2. Receive (server) SYN-ACK (covers options)

Receive_SYN_traffic_key:

```
cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1
```

IPv6/TCP:

```
6e 06 15 20 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cb
19 3c cc ed e0 12 ff ff ea bb 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 71 da ab c8 13 e4 ab 99
1d 10 54 3d dc 28 43 a8 4e 78 a6 bc fd c5 ed 80
```

MAC:

```
dc 28 43 a8 4e 78 a6 bc fd c5 ed 80
```

7.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
61 74 c3 55 7a be d2 75 74 db a3 71 85 f0 03 00
```

IPv6/TCP:

```
6e 04 a7 06 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ed
a6 74 4e cc c0 18 01 00 32 80 00 00 01 01 08 0a
13 e4 ab a3 71 da ab c8 1d 10 3d 54 7b 6a 45 5c
0d 4f 5f 01 83 5b aa b3 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
7b 6a 45 5c 0d 4f 5f 01 83 5b aa b3
```

7.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1
```

IPv6/TCP:

```
6e 06 15 20 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cc
19 3c cd 30 c0 18 01 00 52 f4 00 00 01 01 08 0a
71 da ab d3 13 e4 ab a3 1d 10 54 3d c1 06 9b 7d
fd 3d 69 3a 6d f3 f2 89 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
c1 06 9b 7d fd 3d 69 3a 6d f3 f2 89
```

7.2. AES MAC (omits TCP options)

7.2.1. Send (client) SYN (omits options)

Send_SYN_traffic_key:

```
a9 4f 51 12 63 e4 09 3d 35 dd 81 8c 13 bb bf 53
```

IPv6/TCP:

```
6e 09 3d 76 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4a
00 00 00 00 e0 02 ff ff 75 ff 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 14 27 5b 3b 00 00 00 00
1d 10 3d 54 3d 45 b4 34 2d e8 bb 15 30 84 78 98
```

MAC:

```
3d 45 b4 34 2d e8 bb 15 30 84 78 98
```

7.2.2. Receive (server) SYN-ACK (omits options)

Receive_SYN_traffic_key:

```
92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a
```

IPv6/TCP:

```
6e 0c 60 0a 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 45
b0 1d a7 4b e0 12 ff ff a7 0c 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 17 82 24 5b 14 27 5b 3b
1d 10 54 3d 1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5
```

MAC:

```
1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5
```

7.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
4f b2 08 6e 40 2c 67 90 79 ed 65 d4 bf 97 69 3d
```

IPv6/TCP:

```
6e 09 3d 76 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4b
a6 24 61 46 c0 18 01 00 c3 6d 00 00 01 01 08 0a
14 27 5b 4f 17 82 24 5b 1d 10 3d 54 29 0c f4 14
cc b4 7a 33 32 76 e7 f8 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
29 0c f4 14 cc b4 7a 33 32 76 e7 f8
```

7.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a
```

IPv6/TCP:

```
6e 0c 60 0a 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 46
b0 1d a7 8e c0 18 01 00 34 51 00 00 01 01 08 0a
17 82 24 65 14 27 5b 4f 1d 10 54 3d 99 51 5f fc
d5 40 34 99 f6 19 fd 1b ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
99 51 5f fc d5 40 34 99 f6 19 fd 1b
```

8. Observed Implementation Errors

The following is a partial list of implementation errors that this set of test vectors is intended to validate.

8.1. Algorithm issues

- o Underlying implementation of HMAC SHA1 or AES128 CMAC does not pass their corresponding test vectors [RFC2202] [RFC4493]

- o The SNE algorithm does not consider corner cases (the pseudocode in [RFC5925] was not intended as complete, as discussed in [To21], the latter of which includes its own validation sequence)

8.2. Algorithm parameters

- o KDF context length is incorrect, e.g. it does not include TCP header length + payload length (it should, per 5.2 of TCP-AO [RFC5925])

- o KDF calculation does not start from counter i = 1 (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])

- o KDF calculation does not include output length in bits, contained in two bytes in network byte order (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])

- o KDF uses keys generated from current TCP segment sequence numbers (KDF should use only local and remote ISNs or zero, as indicated in Sec. 5.2 of TCP-AO [RFC5925])

8.3. String handling issues

The strings indicated in TCP-AO and its algorithms are indicated as a sequence of bytes of known length. In some implementations, string lengths are indicated by a terminal value (e.g., zero in C). This terminal value is not included as part of the string for calculations.

- o Password includes the last zero-byte (it should not)

- o Label "TCP-AO" includes the last zero byte (it should not)

8.4. Header coverage issues

- o TCP checksum and/or MAC is not zeroed properly before calculation (both should be)

- o TCP header is not included to the MAC calculation (it should be)

- o TCP options are not included to the MAC calculation by default (there is a separate parameter in the master key tuple to ignore options; this document provides test vectors for both options-included and options-excluded cases)

9. Security Considerations

This document is intended to assist in the validation of implementations of TCP-AO, to further enable its more widespread use as a security mechanism to authenticate not only TCP payload contents but the TCP headers and protocol.

The master_key of "testvector" used here for test vector generation SHOULD NOT be used operationally.

10. IANA Considerations

This document contains no IANA issues. This section should be removed upon publication as an RFC.

11. References

11.1. Normative References

[RFC791] Postel, J., "Internet Protocol," RFC 791, Sept. 1981.

[RFC793] Postel, J., "Transmission Control Protocol," RFC 793, September 1981.

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

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

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

[RFC5926] Lebovitz, G., and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)," RFC 5925, June 2010.

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal," RFC 6978, July 2013.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 2119, May 2017.

[RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.

11.2. Informative References

[RFC2202] Cheng, P., and R. Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1," RFC 2202, Sept. 1997.

[RFC4493] Song, JH, R. Poovendran, J. Lee, T. Iwata, "The AES-CMAC Algorithm," RFC 4493, June 2006.

[To21] Touch, J., "Sequence Number Extension for Windowed Protocols," draft-tsvwg-touch-sne, Apr. 2021.

12. Acknowledgments

(TBD)

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266 USA
Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Juhamatti Kuusisaari
Infinera Corporation
Sinimaentie 6c
FI-02630 Espoo, Finland
Email: jkuusisaari@infinera.com

TCPM
Internet Draft
Intended status: Informational
Expires: September 2022

J. Touch
Independent consultant
J. Kuusisaari
Infinera

March 3, 2022

TCP-AO Test Vectors
draft-ietf-tcpm-ao-test-vectors-09.txt

Abstract

This document provides test vectors to validate implementations of the two mandatory authentication algorithms specified for the TCP Authentication Option over both IPv4 and IPv6. This includes validation of the key derivation function (KDF) based on a set of test connection parameters as well as validation of the message authentication code (MAC). Vectors are provided for both currently required pairs of KDF and MAC algorithms: KDF_HMAC_SHA1 and HMAC-SHA-1-96, and KDF_AES_128_CMAC and AES-128-CMAC-96. The vectors also validate both whole TCP segments as well as segments whose options are excluded for middlebox traversal.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at https://www.ietf.org/shadow.html

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 3, 2022.

Copyright and License Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction....3
2. Conventions used in this document....4
3. Input Test Vectors....4
3.1. TCP Connection Parameters....4
3.1.1. TCP-AO parameters....4
3.1.2. Active (client) side parameters....4
3.1.3. Passive (server) side parameters....5
3.1.4. Other IP fields and options....5
3.1.5. Other TCP fields and options....5
4. IPv4 SHA-1 Output Test Vectors....5
4.1. HMAC-SHA-1-96 (default - covers TCP options)....6
4.1.1. Send (client) SYN (covers options)....6
4.1.2. Receive (server) SYN-ACK (covers options)....6
4.1.3. Send (client) non-SYN (covers options)....7
4.1.4. Receive (server) non-SYN (covers options)....7
4.2. HMAC-SHA-1-96 (omits TCP options)....8
4.2.1. Send (client) SYN (omits options)....8
4.2.2. Receive (server) SYN-ACK (omits options)....8
4.2.3. Send (client) non-SYN (omits options)....9
4.2.4. Receive (server) non-SYN (omits options)....10
5. IPv4 AES-128 Output Test Vectors....10
5.1. AES-128-CMAC-96 (default - covers TCP options)....10
5.1.1. Send (client) SYN (covers options)....10
5.1.2. Receive (server) SYN-ACK (covers options)....11
5.1.3. Send (client) non-SYN (covers options)....12
5.1.4. Receive (server) non-SYN (covers options)....12
5.2. AES-128-CMAC-96 (omits TCP options)....13
5.2.1. Send (client) SYN (omits options)....13
5.2.2. Receive (server) SYN-ACK (omits options)....13

5.2.3. Send (client) non-SYN (omits options).....14
5.2.4. Receive (server) non-SYN (omits options).....14
6. IPv6 SHA-1 Output Test Vectors.....15
6.1. HMAC-SHA-1-96 (default - covers TCP options).....15
6.1.1. Send (client) SYN (covers options).....15
6.1.2. Receive (server) SYN-ACK (covers options).....16
6.1.3. Send (client) non-SYN (covers options).....16
6.1.4. Receive (server) non-SYN (covers options).....17
6.2. HMAC-SHA-1-96 (omits TCP options).....18
6.2.1. Send (client) SYN (omits options).....18
6.2.2. Receive (server) SYN-ACK (omits options).....18
6.2.3. Send (client) non-SYN (omits options).....19
6.2.4. Receive (server) non-SYN (omits options).....19
7. IPv6 AES-128 Output Test Vectors.....20
7.1. AES-128-CMAC-96 (default - covers TCP options).....20
7.1.1. Send (client) SYN (covers options).....20
7.1.2. Receive (server) SYN-ACK (covers options).....21
7.1.3. Send (client) non-SYN (covers options).....21
7.1.4. Receive (server) non-SYN (covers options).....22
7.2. AES-128-CMAC-96 (omits TCP options).....23
7.2.1. Send (client) SYN (omits options).....23
7.2.2. Receive (server) SYN-ACK (omits options).....23
7.2.3. Send (client) non-SYN (omits options).....24
7.2.4. Receive (server) non-SYN (omits options).....24
8. Observed Implementation Errors.....25
8.1. Algorithm issues.....25
8.2. Algorithm parameters.....25
8.3. String handling issues.....26
8.4. Header coverage issues.....26
9. Security Considerations.....26
10. IANA Considerations.....26
11. References.....27
11.1. Normative References.....27
11.2. Informative References.....27
12. Acknowledgments.....28

1. Introduction

This document provides test vectors to validate the correct implementation of the TCP Authentication Option (TCP-AO) [RFC5925] and its mandatory cryptographic algorithms defined in [RFC5926]. It includes the specification of all endpoint parameters to generate the variety of TCP segments covered by different keys and MAC coverage, i.e., both the default case and the variant where TCP options are ignored for middlebox traversal. It also includes both default key derivation functions (KDFs) and MAC generation

algorithms [RFC5926] and lists common pitfalls of implementing the algorithms correctly.

The experimental extension to support NAT traversal [RFC6978] is not included in the provided test vectors.

This document provides test vectors from multiple implementations that have been validated against each other for interoperability.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Input Test Vectors

3.1. TCP Connection Parameters

The following parameters are used throughout this suite of test vectors. The terms 'active' and 'passive' are used as defined for TCP [RFC793].

3.1.1. TCP-AO parameters

The following values are used for all exchanges. This suite does not test key switchover. The KeyIDs are as indicated for TCP-AO [RFC5925]. The Master_Key is used to derive the traffic keys [RFC5926].

Active (client) side KeyID: 61 decimal (0x3d hexadecimal)

Passive (server) side KeyID: 84 decimal (0x54 hexadecimal)

Master_Key: "testvector" (length = 10 bytes)

3.1.2. Active (client) side parameters

The following endpoint parameters are used on the active side of the TCP connection, i.e., the side that initiates the TCP SYN.

For IPv4: 10.11.12.13 (dotted decimal)

For IPv6: fd00::1 (IPv6 hexadecimal)

TCP port: (varies)

3.1.3. Passive (server) side parameters

The following endpoint parameters are used for the passive side of the TCP connection, i.e., the side that responds with a TCP SYN-ACK.

For IPv4: 172.27.28.29 (dotted decimal)

For IPv6: fd00::2 (IPv6 hexadecimal)

TCP port = 179 decimal (BGP)

3.1.4. Other IP fields and options

No IP options are used in these test vectors.

All IPv4 packets use the following other parameters [RFC791]: DSCP = 111000 binary (CS7) as is typical for BGP, ECN = 00 binary, set DF, and clear MF.

IPv4 uses a TTL of 255 decimal; IPv6 uses a hop limit of 255 decimal.

All IPv6 packets use the following other parameters [RFC8200]: traffic class = 0xe0 hexadecimal (DSCP = 111000 binary CS7, as is typical for BGP, with ECN = 00 binary) and no EHs.

3.1.5. Other TCP fields and options

The SYN and SYN-ACK segments include MSS [RFC793], NOP, WindowScale [RFC7323], SACK Permitted [RFC2018], TimeStamp [RFC7323], and TCP-AO [RFC5925], in that order.

All other example segments include NOP, NOP, TimeStamp, and TCP-AO, in that order.

All segment URG pointers are zero [RFC793]. All segments with data set the PSH flag [RFC793].

Each TCP connection below uses the Initial Sequence Numbers (ISNs) as indicated at the front of each corresponding section.

4. IPv4 SHA-1 Output Test Vectors

The SHA-1 KDF and MAC algorithms, KDF_HMAC_SHA1 and HMAC-SHA-1-96, are computed as specified for TCP-AO [RFC5926].

In the following sections, all values are indicated as 2-digit hexadecimal values with spacing per line representing the contents of 16 consecutive bytes, as is typical for data dumps. The IP/TCP data indicates the entire IP packet, including the TCP segment and its options (whether covered by TCP-AO or not, as indicated), including TCP-AO.

4.1. HMAC-SHA-1-96 (default - covers TCP options)

4.1.1. Send (client) SYN (covers options)

Client ISN = 0xfbfbab5a

Send_SYN_traffic_key:

```
6d 63 ef 1b 02 fe 15 09 d4 b1 40 27 07 fd 7b 04
16 ab b7 4f
```

IPv4/TCP:

```
45 e0 00 4c dd 0f 40 00 ff 06 bf 6b 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5a 00 00 00 00
e0 02 ff ff ca c4 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 15 5a b7 00 00 00 00 1d 10 3d 54
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

MAC:

```
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

4.1.2. Receive (server) SYN-ACK (covers options)

Server ISN = 0x11c14261

Receive_SYN_traffic_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 4c 65 06 40 00 ff 06 37 75 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 61 fb fb ab 5b
e0 12 ff ff 37 76 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 84 a5 0b eb 00 15 5a b7 1d 10 54 3d
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

MAC:

```
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

4.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
d2 e5 9c 65 ff c7 b1 a3 93 47 65 64 63 b7 0e dc
24 a1 3d 71
```

IPv4/TCP:

```
45 e0 00 87 36 a1 40 00 ff 06 65 9f 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5b 11 c1 42 62
c0 18 01 04 a1 62 00 00 01 01 08 0a 00 15 5a c1
84 a5 0b eb 1d 10 3d 54 70 64 cf 99 8c c6 c3 15
c2 c2 e2 bf ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
70 64 cf 99 8c c6 c3 15 c2 c2 e2 bf
```

4.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 87 1f a9 40 00 ff 06 7c 97 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 62 fb fb ab 9e
c0 18 01 00 40 0c 00 00 01 01 08 0a 84 a5 0b f5
00 15 5a c1 1d 10 54 3d a6 3f 0e cb bb 2e 63 5c
95 4d ea c7 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
a6 3f 0e cb bb 2e 63 5c 95 4d ea c7
```

4.2. HMAC-SHA-1-96 (omits TCP options)

4.2.1. Send (client) SYN (omits options)

Client ISN = 0xcb0efbee

Send_SYN_traffic_key:

```
30 ea a1 56 0c f0 be 57 da b5 c0 45 22 9f b1 0a
42 3c d7 ea
```

IPv4/TCP:

```
45 e0 00 4c 53 99 40 00 ff 06 48 e2 0a 0b 0c 0d
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ee 00 00 00 00
e0 02 ff ff 54 1f 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 02 4c ce 00 00 00 00 1d 10 3d 54
80 af 3c fe b8 53 68 93 7b 8f 9e c2
```

MAC:

```
80 af 3c fe b8 53 68 93 7b 8f 9e c2
```

4.2.2. Receive (server) SYN-ACK (omits options)

Server ISN = 0xacd5b5e1

Receive_SYN_traffic_key:

```
b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41a
01 a8 30 7f
```

IPv4/TCP:

```
45 e0 00 4c 32 84 40 00 ff 06 69 f7 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e1 cb 0e fb ef
e0 12 ff ff 38 8e 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 57 67 72 f3 00 02 4c ce 1d 10 54 3d
09 30 6f 9a ce a6 3a 8c 68 cb 9a 70
```

MAC:

```
09 30 6f 9a ce a6 3a 8c 68 cb 9a 70
```

4.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
f3 db 17 93 d7 91 0e cd 80 6c 34 f1 55 ea 1f 00
34 59 53 e3
```

IPv4/TCP:

```
45 e0 00 87 a8 f5 40 00 ff 06 f3 4a 0a 0b 0c 0d
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ef ac d5 b5 e2
c0 18 01 04 6c 45 00 00 01 01 08 0a 00 02 4c ce
57 67 72 f3 1d 10 3d 54 71 06 08 cc 69 6c 03 a2
71 c9 3a a5 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
71 06 08 cc 69 6c 03 a2 71 c9 3a a5
```

4.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41
01 a8 30 7f
```

IPv4/TCP:

```
45 e0 00 87 54 37 40 00 ff 06 48 09 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e2 cb 0e fc 32
c0 18 01 00 46 b6 00 00 01 01 08 0a 57 67 72 f3
00 02 4c ce 1d 10 54 3d 97 76 6e 48 ac 26 2d e9
ae 61 b4 f9 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
97 76 6e 48 ac 26 2d e9 ae 61 b4 f9
```

5. IPv4 AES-128 Output Test Vectors

The AES-128 KDF and MAC algorithms, KDF_AES_128_CMAC and AES-128-CMAC-96, are computed as specified for TCP-AO [RFC5926].

In the following sections, all values are indicated as 2-digit hexadecimal values with spacing per line representing the contents of 16 consecutive bytes, as is typical for data dumps. The IP/TCP data indicates the entire IP packet, including the TCP segment and its options (whether covered by TCP-AO or not, as indicated), including TCP-AO.

5.1. AES-128-CMAC-96 (default - covers TCP options)

5.1.1. Send (client) SYN (covers options)

Client ISN = 0x787a1ddf

Send_SYN_traffic_key:

```
f5 b8 b3 d5 f3 4f db b6 eb 8d 4a b9 66 0e 60 e3
```

IP/TCP:

```
45 e0 00 4c 7b 9f 40 00 ff 06 20 dc 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d df 00 00 00 00
e0 02 ff ff 5a 0f 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 7e d0 00 00 00 00 1d 10 3d 54
e4 77 e9 9c 80 40 76 54 98 e5 50 91
```

MAC:

```
e4 77 e9 9c 80 40 76 54 98 e5 50 91
```

5.1.2. Receive (server) SYN-ACK (covers options)

Server ISN = 0xfadd6de9

Receive_SYN_traffic_key:

```
4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1
```

IPv4/TCP:

```
45 e0 00 4c 4b ad 40 00 ff 06 50 ce ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d e9 78 7a 1d e0
e0 12 ff ff f3 f2 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 93 f4 e9 e8 00 01 7e d0 1d 10 54 3d
d6 ad a7 bc 4c dd 53 6d 17 69 db 5f
```

MAC:

```
d6 ad a7 bc 4c dd 53 6d 17 69 db 5f
```

5.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
8c 8a e0 e8 37 1e c5 cb b9 7e a7 9d 90 41 83 91
```

IPv4/TCP:

```
45 e0 00 87 fb 4f 40 00 ff 06 a0 f0 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d e0 fa dd 6d ea
c0 18 01 04 95 05 00 00 01 01 08 0a 00 01 7e d0
93 f4 e9 e8 1d 10 3d 54 77 41 27 42 fa 4d c4 33
ef f0 97 3e ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
77 41 27 42 fa 4d c4 33 ef f0 97 3e
```

5.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1
```

IPv4/TCP:

```
45 e0 00 87 b9 14 40 00 ff 06 e3 2b ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d ea 78 7a 1e 23
c0 18 01 00 e7 db 00 00 01 01 08 0a 93 f4 e9 e8
00 01 7e d0 1d 10 54 3d f6 d9 65 a7 83 82 a7 48
45 f7 2d ac ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
f6 d9 65 a7 83 82 a7 48 45 f7 2d ac
```

5.2. AES-128-CMAC-96 (omits TCP options)

5.2.1. Send (client) SYN (omits options)

Client ISN = 0x389bed71

Send_SYN_traffic_key:

```
2c db ae 13 92 c4 94 49 fa 92 c4 50 97 35 d5 0e
```

IPv4/TCP:

```
45 e0 00 4c f2 2e 40 00 ff 06 aa 4c 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 71 00 00 00 00
e0 02 ff ff 70 bf 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 85 e1 00 00 00 00 1d 10 3d 54
c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49
```

MAC:

```
c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49
```

5.2.2. Receive (server) SYN-ACK (omits options)

Server ISN = 0xd3844a6f

Receive_SYN_traffic_key:

```
3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a
```

IPv4/TCP:

```
45 e0 00 4c 6c c0 40 00 ff 06 2f bb ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 6f 38 9b ed 72
e0 12 ff ff e4 45 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a ce 45 98 38 00 01 85 e1 1d 10 54 3d
3a 6a bb 20 7e 49 b1 be 71 36 db 90
```

MAC:

```
3a 6a bb 20 7e 49 b1 be 71 36 db 90
```

5.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
03 5b c4 00 a3 41 ff e5 95 f5 9f 58 00 50 06 ca
```

IPv4/TCP:

```
45 e0 00 87 ee 91 40 00 ff 06 ad ae 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 72 d3 84 4a 70
c0 18 01 04 88 51 00 00 01 01 08 0a 00 01 85 e1
ce 45 98 38 1d 10 3d 54 75 85 e9 e9 d5 c3 ec 85
7b 96 f8 37 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
75 85 e9 e9 d5 c3 ec 85 7b 96 f8 37
```

5.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a
```

IPv4/TCP:

```
45 e0 00 87 6a 21 40 00 ff 06 32 1f ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 70 38 9b ed 72
c0 18 01 00 04 49 00 00 01 01 08 0a ce 45 98 38
00 01 85 e1 1d 10 54 3d 5c 04 0f d9 23 33 04 76
5c 09 82 f4 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
5c 04 0f d9 23 33 04 76 5c 09 82 f4
```

6. IPv6 SHA-1 Output Test Vectors

The SHA-1 KDF and MAC algorithms, KDF_HMAC_SHA1 and HMAC-SHA-1-96, are computed as specified for TCP-AO [RFC5926].

6.1. HMAC-SHA-1-96 (default - covers TCP options)

6.1.1. Send (client) SYN (covers options)

Client ISN = 0x176a833f

Send_SYN_traffic_key:

```
62 5e c0 9d 57 58 36 ed c9 b6 42 84 18 bb f0 69
89 a3 61 bb
```

IPv6/TCP:

```
6e 08 91 dc 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 3f
00 00 00 00 e0 02 ff ff 47 21 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 41 d0 87 00 00 00 00
1d 10 3d 54 90 33 ec 3d 73 34 b6 4c 5e dd 03 9f
```

MAC:

```
90 33 ec 3d 73 34 b6 4c 5e dd 03 9f
```

6.1.2. Receive (server) SYN-ACK (covers options)

Server ISN = 0x3f51994b

Receive_SYN_traffic_key:

```
e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7
71 eb cb 4a
```

IPv6/TCP:

```
6e 01 00 9e 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4b
17 6a 83 40 e0 12 ff ff bf ec 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a bd 33 12 9b 00 41 d0 87
1d 10 54 3d f1 cb a3 46 c3 52 61 63 f7 1f 1f 55
```

MAC:

```
f1 cb a3 46 c3 52 61 63 f7 1f 1f 55
```

6.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
1e d8 29 75 f4 ea 44 4c 61 58 0c 5b d9 0d bd 61
bb c9 1b 7e
```

IPv6/TCP:

```
6e 08 91 dc 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 40
3f 51 99 4c c0 18 01 00 32 9c 00 00 01 01 08 0a
00 41 d0 91 bd 33 12 9b 1d 10 3d 54 bf 08 05 fe
b4 ac 7b 16 3d 6f cd f2 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
bf 08 05 fe b4 ac 7b 16 3d 6f cd f2
```

6.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7
71 eb cb 4a
```

IPv6/TCP:

```
6e 01 00 9e 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4c
17 6a 83 83 c0 18 01 00 ee 6e 00 00 01 01 08 0a
bd 33 12 a5 00 41 d0 91 1d 10 54 3d 6c 48 12 5c
11 33 5b ab 9a 07 a7 97 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
6c 48 12 5c 11 33 5b ab 9a 07 a7 97
```

6.2. HMAC-SHA-1-96 (omits TCP options)

6.2.1. Send (client) SYN (omits options)

Client ISN = 0x020c1e69

Send_SYN_traffic_key:

```
31 a3 fa f6 9e ff ae 52 93 1b 7f 84 54 67 31 5c
27 0a 4e dc
```

IPv6/TCP:

```
6e 07 8f cd 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 69
00 00 00 00 e0 02 ff ff a4 1a 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 9d b9 5b 00 00 00 00
1d 10 3d 54 88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

MAC:

```
88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

6.2.2. Receive (server) SYN-ACK (omits options)

Server ISN = 0xeba3734d

Receive_SYN_traffic_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

IPv6/TCP:

```
6e 0a 7e 1f 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4d
02 0c 1e 6a e0 12 ff ff 77 4d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 5e c9 9b 70 00 9d b9 5b
1d 10 54 3d 3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

MAC:

```
3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

6.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
b3 4e ed 6a 93 96 a6 69 f1 c4 f4 f5 76 18 f3 65
6f 52 c7 ab
```

IPv6/TCP:

```
6e 07 8f cd 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 6a
eb a3 73 4e c0 18 01 00 83 e6 00 00 01 01 08 0a
00 9d b9 65 5e c9 9b 70 1d 10 3d 54 48 bd 09 3b
19 24 e0 01 19 2f 5b f0 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
48 bd 09 3b 19 24 e0 01 19 2f 5b f0
```

6.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

IPv6/TCP:

```
6e 0a 7e 1f 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4e
02 0c 1e ad c0 18 01 00 71 6a 00 00 01 01 08 0a
5e c9 9b 7a 00 9d b9 65 1d 10 54 3d 55 9a 81 94
45 b4 fd e9 8d 9e 13 17 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
55 9a 81 94 45 b4 fd e9 8d 9e 13 17
```

7. IPv6 AES-128 Output Test Vectors

The AES-128 KDF and MAC algorithms, KDF_AES_128_CMAC and AES-128-CMAC-96, are computed as specified for TCP-AO [RFC5926].

7.1. AES-128-CMAC-96 (default - covers TCP options)

7.1.1. Send (client) SYN (covers options)

Client ISN = 0x193cccec

Send_SYN_traffic_key:

```
fa 5a 21 08 88 2d 39 d0 c7 19 29 17 5a b1 b7 b8
```

IP/TCP:

```
6e 04 a7 06 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ec
00 00 00 00 e0 02 ff ff de 5d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 13 e4 ab 99 00 00 00 00
1d 10 3d 54 59 b5 88 10 74 81 ac 6d c3 92 70 40
```

MAC:

```
59 b5 88 10 74 81 ac 6d c3 92 70 40
```

7.1.2. Receive (server) SYN-ACK (covers options)

Server ISN = 0xa6744ecb

Receive_SYN_traffic_key:

```
cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1
```

IPv6/TCP:

```
6e 06 15 20 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cb
19 3c cc ed e0 12 ff ff ea bb 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 71 da ab c8 13 e4 ab 99
1d 10 54 3d dc 28 43 a8 4e 78 a6 bc fd c5 ed 80
```

MAC:

```
dc 28 43 a8 4e 78 a6 bc fd c5 ed 80
```

7.1.3. Send (client) non-SYN (covers options)

Send_other_traffic_key:

```
61 74 c3 55 7a be d2 75 74 db a3 71 85 f0 03 00
```

IPv6/TCP:

```
6e 04 a7 06 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ed
a6 74 4e cc c0 18 01 00 32 80 00 00 01 01 08 0a
13 e4 ab a3 71 da ab c8 1d 10 3d 54 7b 6a 45 5c
0d 4f 5f 01 83 5b aa b3 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
7b 6a 45 5c 0d 4f 5f 01 83 5b aa b3
```

7.1.4. Receive (server) non-SYN (covers options)

Receive_other_traffic_key:

```
cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1
```

IPv6/TCP:

```
6e 06 15 20 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cc
19 3c cd 30 c0 18 01 00 52 f4 00 00 01 01 08 0a
71 da ab d3 13 e4 ab a3 1d 10 54 3d c1 06 9b 7d
fd 3d 69 3a 6d f3 f2 89 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
c1 06 9b 7d fd 3d 69 3a 6d f3 f2 89
```

7.2. AES-128-CMAC-96 (omits TCP options)

7.2.1. Send (client) SYN (omits options)

Client ISN = 0xb01da74a

Send_SYN_traffic_key:

```
a9 4f 51 12 63 e4 09 3d 35 dd 81 8c 13 bb bf 53
```

IPv6/TCP:

```
6e 09 3d 76 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4a
00 00 00 00 e0 02 ff ff 75 ff 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 14 27 5b 3b 00 00 00 00
1d 10 3d 54 3d 45 b4 34 2d e8 bb 15 30 84 78 98
```

MAC:

```
3d 45 b4 34 2d e8 bb 15 30 84 78 98
```

7.2.2. Receive (server) SYN-ACK (omits options)

Server ISN = 0xa6246145

Receive_SYN_traffic_key:

```
92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a
```

IPv6/TCP:

```
6e 0c 60 0a 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 45
b0 1d a7 4b e0 12 ff ff a7 0c 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 17 82 24 5b 14 27 5b 3b
1d 10 54 3d 1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5
```

MAC:

```
1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5
```

7.2.3. Send (client) non-SYN (omits options)

Send_other_traffic_key:

```
4f b2 08 6e 40 2c 67 90 79 ed 65 d4 bf 97 69 3d
```

IPv6/TCP:

```
6e 09 3d 76 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4b
a6 24 61 46 c0 18 01 00 c3 6d 00 00 01 01 08 0a
14 27 5b 4f 17 82 24 5b 1d 10 3d 54 29 0c f4 14
cc b4 7a 33 32 76 e7 f8 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
29 0c f4 14 cc b4 7a 33 32 76 e7 f8
```

7.2.4. Receive (server) non-SYN (omits options)

Receive_other_traffic_key:

```
92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a
```

IPv6/TCP:

```
6e 0c 60 0a 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 46
b0 1d a7 8e c0 18 01 00 34 51 00 00 01 01 08 0a
17 82 24 65 14 27 5b 4f 1d 10 54 3d 99 51 5f fc
d5 40 34 99 f6 19 fd 1b ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
99 51 5f fc d5 40 34 99 f6 19 fd 1b
```

8. Observed Implementation Errors

The following is a partial list of implementation errors that this set of test vectors is intended to validate.

8.1. Algorithm issues

- Underlying implementation of HMAC-SHA-1-96 or AES-128-CMAC-96 does not pass their corresponding test vectors [RFC2202] [RFC4493]

- The SNE algorithm does not consider corner cases, possibly because the pseudocode in [RFC5925] was not intended as complete, as discussed in [RFC9187], the latter of which includes its own validation sequence.

8.2. Algorithm parameters

- KDF context length is incorrect, e.g., it does not include TCP header length + payload length (it should, per 5.2 of TCP-AO [RFC5925])

- KDF calculation does not start from counter i = 1 (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])

- KDF calculation does not include output length in bits, contained in two bytes in network byte order (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])

- o KDF uses keys generated from current TCP segment sequence numbers (KDF should use only local and remote ISNs or zero, as indicated in Sec. 5.2 of TCP-AO [RFC5925])

8.3. String handling issues

The strings indicated in TCP-AO and its algorithms are indicated as a sequence of bytes of known length. In some implementations, string lengths are indicated by a terminal value (e.g., zero in C). This terminal value is not included as part of the string for calculations.

- o Password includes the last zero-byte (it should not)

- o Label "TCP-AO" includes the last zero byte (it should not)

8.4. Header coverage issues

- o TCP checksum and/or MAC is not zeroed properly before calculation (both should be)

- o TCP header is not included in the MAC calculation (it should be)

- o TCP options are not included in the MAC calculation by default.

There is a separate parameter in the Master Key Tuple (MKT) [RFC5925] to ignore options; this document provides test vectors for both options-included and options-excluded cases.

9. Security Considerations

This document is intended to assist in the validation of implementations of TCP-AO, to further enable its more widespread use as a security mechanism to authenticate not only TCP payload contents but the TCP headers and protocol.

The Master_Key of "testvector" used here for test vector generation SHOULD NOT be used operationally.

10. IANA Considerations

This document contains no IANA issues. This section should be removed upon publication as an RFC.

11. References

11.1. Normative References

[RFC791] Postel, J., "Internet Protocol," RFC 791, Sept. 1981.

[RFC793] Postel, J., "Transmission Control Protocol," RFC 793, September 1981.

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

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

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

[RFC5926] Lebovitz, G., and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)," RFC 5926, June 2010.

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal," RFC 6978, July 2013.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 8174, May 2017.

[RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.

11.2. Informative References

[RFC2202] Cheng, P., and R. Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1," RFC 2202, Sept. 1997.

[RFC4493] Song, JH, R. Poovendran, J. Lee, T. Iwata, "The AES-CMAC Algorithm," RFC 4493, June 2006.

[RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols," RFC 9187, Jan. 2022.

12. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266 USA
Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Juhamatti Kuusisaari
Infinera Corporation
Sinimaentie 6c
FI-02630 Espoo, Finland
Email: jkuusisaari@infinera.com

TCP Maintenance Working Group
Internet-Draft
Obsoletes: 6937 (if approved)
Intended status: Standards Track
Expires: 26 August 2021

M. Mathis
N. Dukkipati
Y. Cheng
Google, Inc.

22 February 2021

Proportional Rate Reduction for TCP
draft-ietf-tcpm-prr-rfc6937bis-01

Abstract

This document updates the experimental Proportional Rate Reduction (PRR) algorithm, described RFC 6937, to standards track. PRR potentially replaces the Fast Recovery and Rate-Halving algorithms. All of these algorithms regulate the amount of data sent by TCP or other transport protocol during loss recovery. PRR accurately regulates the actual flight size through recovery such that at the end of recovery it will be as close as possible to the ssthresh, as determined by the congestion control algorithm.

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 https://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 26 August 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
1.1. Document and WG Information 3
2. Background 3
3. Changes From RFC 6937 5
4. Relationships to other standards 6
5. Definitions 7
6. Algorithms 8
7. Examples 9
8. Properties 12
9. Adapting PRR to other transport protocols 14
10. Acknowledgements 14
11. Security Considerations 15
12. Normative References 15
13. Informative References 15
Appendix A. Strong Packet Conservation Bound 17
Authors' Addresses 18

1. Introduction

This document updates the Proportional Rate Reduction (PRR) algorithm described in [RFC6937] from experimental to standards track. PRR accuracy regulates the amount of data sent during loss recovery, such that at the end of recovery the flight size will be as close as possible to the ssthresh, as determined by the congestion control algorithm. PRR has been deployed in at least 3 major operating systems covering the vast majority of today's web traffic.

The only change from RFC 6937 is the introduction of a new heuristic that replaces a manual configuration parameter. There have been no changes to the behaviors of the algorithms or the previously published results. The new heuristic only changes behaviors in corner cases that were not relevant prior to the Lost Retransmission Detection (LRD) algorithm which was not implemented until after RFC 6937 was published. This document also includes additional discussion about integration into other congestion control and recovery algorithms.

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]

1.1. Document and WG Information

Formatted: 2021-02-22 14:22:57-08:00

Please send all comments, questions and feedback to tcpm@ietf.org

About revision 00:

The introduction above was drawn from draft-mathis-tcpm-rfc6937bis-00. All of the text below was copied verbatim from RFC 6937, to facilitate comparison between RFC 6937 and this document as it evolves.

About revision 01:

* Recast the RFC 6937 introduction as background

* Made "Changes From RFC 6937" an explicit section

* Made Relationships to other standards more explicit

* Added a generalized safeACK heuristic

* Provided hints for non TCP implementations

* Added language about detecting ACK splitting, but have no advice on actions (yet)

2. Background

This section is copied almost verbatim from the introduction to RFC 6937.

Standard congestion control [RFC5681] requires that TCP (and other protocols) reduce their congestion window (cwnd) in response to losses. Fast Recovery, described in the same document, is the reference algorithm for making this adjustment. Its stated goal is to recover TCP's self clock by relying on returning ACKs during recovery to clock more data into the network. Fast Recovery typically adjusts the window by waiting for one half round-trip time (RTT) of ACKs to pass before sending any data. It is fragile because it cannot compensate for the implicit window reduction caused by the losses themselves.

RFC 6675 [RFC6675] makes Fast Recovery with Selective Acknowledgement (SACK) [RFC2018] more accurate by computing "pipe", a sender side estimate of the number of bytes still outstanding in the network.

With RFC 6675, Fast Recovery is implemented by sending data as necessary on each ACK to prevent pipe from falling below slow-start threshold (ssthresh), the window size as determined by the congestion control algorithm. This protects Fast Recovery from timeouts in many cases where there are heavy losses, although not if the entire second half of the window of data or ACKs are lost. However, a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a burst of data.

The Rate-Halving algorithm sends data on alternate ACKs during recovery, such that after 1 RTT the window has been halved. Rate-Halving was implemented in Linux after only being informally published [RHweb], including an uncompleted document [RHID]. Rate-Halving also does not adequately compensate for the implicit window reduction caused by the losses and assumes a net 50% window reduction, which was completely standard at the time it was written but not appropriate for modern congestion control algorithms, such as CUBIC [CUBIC], which reduce the window by less than 50%. As a consequence, Rate-Halving often allows the window to fall further than necessary, reducing performance and increasing the risk of timeouts if there are additional losses.

PRR avoids these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size as determined by the congestion control algorithm. It is patterned after Rate-Halving, but using the fraction that is appropriate for the target window chosen by the congestion control algorithm. During PRR, one of two additional Reduction Bound algorithms limits the total window reduction due to all mechanisms, including transient application stalls and the losses themselves.

We describe two slightly different Reduction Bound algorithms: Conservative Reduction Bound (CRB), which is strictly packet conserving; and a Slow Start Reduction Bound (SSRB), which is more aggressive than CRB by, at most, 1 segment per ACK. PRR-CRB meets the Strong Packet Conservation Bound described in Appendix A; however, in real networks it does not perform as well as the algorithms described in RFC 6675, which prove to be more aggressive in a significant number of cases. SSRB offers a compromise by allowing TCP to send 1 additional segment per ACK relative to CRB in some situations. Although SSRB is less aggressive than RFC 6675 (transmitting fewer segments or taking more time to transmit them), it outperforms it, due to the lower probability of additional losses during recovery.

The Strong Packet Conservation Bound on which PRR and both Reduction Bounds are based is patterned after Van Jacobson's packet conservation principle: segments delivered to the receiver are used as the clock to trigger sending the same number of segments back into the network. As much as possible, PRR and the Reduction Bound algorithms rely on this self clock process, and are only slightly affected by the accuracy of other estimators, such as pipe [RFC6675] and cwnd. This is what gives the algorithms their precision in the presence of events that cause uncertainty in other estimators.

The original definition of the packet conservation principle [Jacobson88] treated packets that are presumed to be lost (e.g., marked as candidates for retransmission) as having left the network. This idea is reflected in the pipe estimator defined in RFC 6675 and used here, but it is distinct from the Strong Packet Conservation Bound as described in Appendix A, which is defined solely on the basis of data arriving at the receiver.

3. Changes From RFC 6937

The largest change since RFC 6937 [RFC6937] is the introduction of a new heuristic that uses good recovery progress (For TCP, snd.una advances and no additional segments are marked as lost) to select which Reduction Bound. RFC 6937 left the choice of Reduction Bound to the discretion of the implementer but recommended to use BBR-SSRB by default. For all of the environments explored in earlier PRR research, the new heuristic is consistent with the old recommendation.

The paper "An Internet-Wide Analysis of Traffic Policing" [Flach2016policing] uncovered a crucial situation, not previously explored, where both Reduction Bounds perform very poorly, but for different reasons. Under many configurations, token bucket traffic policers [token_bucket] can suddenly start discarding a large fraction of the traffic, without any warning to the end systems. The transport congestion control has no opportunity to measure the token rate, and sets ssthresh based on the previously observed path performance. This value for ssthresh may result in a data rate that is substantially larger than the token rate, causing persistent high loss. Under these conditions, both reduction bounds perform very poorly. PRR-CRB is too timid, sometimes causing very long recovery times at smaller than necessary windows, and PRR-SSRB is too aggressive, often causing many retransmissions to be lost multiple times.

Investigating these environments led to the development of a "safeACK" heuristic to dynamically switch between Reduction Bounds: use PRR-SSRB for ACKs reporting that the recovery is making good progress (snd.una is advancing without any new losses) and PRR-CRB otherwise

This heuristic is only invoked where application-limited behavior, losses or other events cause the flight size to fall below ssthresh. The extreme loss rates that make the heuristic important are only common in the presence of token bucket policers, which are pathologically wasteful and inefficient [Flach2016policing]. In these environments the heuristic serves to salvage a bad situation and any reasonable implementation of the heuristic performs far better than either bound by itself. The heuristic has no effect whatsoever in congestion events where there are no lost retransmissions, including all of the examples described below and in RFC 6937.

Since RFC 6937 was written, PRR has also been adapted to perform multiplicative window reduction for non-loss based congestion control algorithms, such as for RFC 3168 style ECN. This is typically done by using some parts of the loss recovery state machine (in particular the RecoveryPoint from RFC 6675) to invoke the PRR ACK processing for exactly one round trip worth of ACKs.

For RFC 6937 we published a companion paper [IMC11] in which we evaluated Fast Retransmit, Rate-Halving and various experimental PRR versions in a large scale measurement study. Today, the legacy algorithms used in that study have already faded from the code base, making such comparisons impossible without recreating historical algorithms. Readers interested in the measurement study should review section 5 of RFC 6937 and the IMC paper [IMC11].

4. Relationships to other standards

PRR is described as modifications to "TCP Congestion Control" [RFC5681], and "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP" [RFC6675]. It is most accurate and more easily implemented with SACK [RFC2018], but does not require SACK.

The SafeACK heuristic came about as a consequence of robust Lost Retransmission Detection under development in an early precursor to [RACK]. Without LRD, policers that cause very high loss rates are guaranteed to also cause retransmission timeouts because both RFC 5681 and RFC 6675 will send retransmissions above the policed rate. PRR and the SafeACK heuristic were already well in place before the RACK algorithm was fully matured. Note that there is no experience implementing or testing RACK without PRR.

For this reason it is recommended that PRR is implemented with RACK.

5. Definitions

The following terms, parameters, and state variables are used as they are defined in earlier documents:

RFC 793: snd.una (send unacknowledged).

RFC 5681: duplicate ACK, FlightSize, Sender Maximum Segment Size (SMSS).

RFC 6675: covered (as in "covered sequence numbers").

Voluntary window reductions: choosing not to send data in response to some ACKs, for the purpose of reducing the sending window size and data rate.

We define some additional variables:

SACKd: The total number of bytes that the scoreboard indicates have been delivered to the receiver. This can be computed by scanning the scoreboard and counting the total number of bytes covered by all sack blocks. If SACK is not in use, SACKd is not defined.

DeliveredData: The total number of bytes that the current ACK indicates have been delivered to the receiver. With SACK, DeliveredData can be computed precisely as the change in snd.una, plus the (signed) change in SACKd. In recovery without SACK, DeliveredData is estimated to be 1 SMSS on duplicate acknowledgements, and on a subsequent partial or full ACK, DeliveredData is estimated to be the change in snd.una, minus 1 SMSS for each preceding duplicate ACK. If this calculation results in a negative DeliveredData the data sender can infer that the receiver is using a ACK splitting attack [and do what? @@@@]

Note that DeliveredData is robust; for TCP using SACK, DeliveredData can be precisely computed anywhere along the return path by inspecting the returning ACKs. The consequence of missing ACKs is

that later ACKs will show a larger DeliveredData. Furthermore, for any TCP (with or without SACK), the sum of DeliveredData must agree with the forward progress over the same time interval.

safeACK: A local variable indicating that the current ACK reported good progress -- snd.una advanced with no additional segments newly marked lost.

sndcnt: A local variable indicating exactly how many bytes should be sent in response to each ACK. Note that the decision of which data to send (e.g., retransmit missing data or send more new data) is out of scope for this document.

6. Algorithms

At the beginning of recovery, initialize PRR state. This assumes a modern congestion control algorithm, CongCtrlAlg(), that might set ssthresh to something other than FlightSize/2:

```
ssthresh = CongCtrlAlg()  // Target cwnd after recovery
prr_delivered = 0         // Total bytes delivered during recovery
prr_out = 0               // Total bytes sent during recovery
RecoverFS = snd.nxt-snd.una // FlightSize at the start of recovery
```

Figure 1

```
On every ACK during recovery compute:
   DeliveredData = change_in(snd.una) + change_in(SACKd)
   prr_delivered += DeliveredData
   pipe = (RFC 6675 pipe algorithm)
   safeACK = (snd.una advances with no new losses)
   if (pipe > ssthresh) {
      // Proportional Rate Reduction
      sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
   } else {
      // Two version of the reduction bound
      if (safeACK) {    // PRR+SSRB
        limit = MAX(prr_delivered - prr_out, DeliveredData) + MSS
      } else {               // PRR+CRB
        limit = prr_delivered - prr_out
      }
      // Attempt to catch up, as permitted by limit
      sndcnt = MIN(ssthresh - pipe, limit)
   }
```

Figure 2

```
On any data transmission or retransmission:
  prr_out += (data sent) // strictly less than or equal to sndcnt
```

Figure 3

7. Examples

We illustrate these algorithms by showing their different behaviors for two scenarios: TCP experiencing either a single loss or a burst of 15 consecutive losses. In all cases we assume bulk data (no application pauses), standard Additive Increase Multiplicative Decrease (AIMD) congestion control, and cwnd = FlightSize = pipe = 20 segments, so ssthresh will be set to 10 at the beginning of recovery. We also assume standard Fast Retransmit and Limited Transmit [RFC3042], so TCP will send 2 new segments followed by 1 retransmit in response to the first 3 duplicate ACKs following the losses.

Each of the diagrams below shows the per ACK response to the first round trip for the various recovery algorithms when the zeroth segment is lost. The top line indicates the transmitted segment number triggering the ACKs, with an X for the lost segment. "cwnd" and "pipe" indicate the values of these algorithms after processing each returning ACK. "Sent" indicates how much 'N'ew or 'R'etransmitted data would be sent. Note that the algorithms for deciding which data to send are out of scope of this document.

We are including the Linux Rate_Halving implementation to illustrate the state-of-the-art at the time, even though this algorithm is no longer supported.

When there is a single loss, PRR with either of the Reduction Bound algorithms has the same behavior. We show "RB", a flag indicating which Reduction Bound subexpression ultimately determined the value of sndcnt. When there are minimal losses, "limit" (both algorithms) will always be larger than ssthresh - pipe, so the sndcnt will be ssthresh - pipe, indicated by "s" in the "RB" row.

```
RFC 6675
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
pipe:    19 19 18 18 17 16 15 14 13 12 11 10 10 10 10 10 10 10 10
sent:     N  N  R                         N  N  N  N  N  N  N  N



Rate-Halving (Historical Linux)
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 11
pipe:    19 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 11 10
sent:     N  N  R     N     N     N     N     N     N     N     N



PRR
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
pipe:    19 19 18 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 10
sent:     N  N  R     N     N     N     N     N     N        N  N
RB:                                                           s  s
    Cwnd is not shown because PRR does not use it.



Key for RB
s: sndcnt = ssthresh - pipe                 // from ssthresh
b: sndcnt = prr_delivered - prr_out + SMSS  // from banked
d: sndcnt = DeliveredData + SMSS            // from DeliveredData
(Sometimes, more than one applies.)
```

Figure 4

Note that all 3 algorithms send the same total amount of data. RFC 6675 experiences a "half window of silence", while the Rate-Halving and PRR spread the voluntary window reduction across an entire RTT.

Next, we consider the same initial conditions when the first 15 packets (0-14) are lost. During the remainder of the lossy RTT, only 5 ACKs are returned to the sender. We examine each of these algorithms in succession.

```
RFC 6675
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X 15 16 17 18 19
cwnd:                                              20 20 11 11 11
pipe:                                              19 19  4 10 10
sent:                                               N  N 7R  R  R



Rate-Halving (Historical Linux)
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X 15 16 17 18 19
cwnd:                                              20 20  5  5  5
pipe:                                              19 19  4  4  4
sent:                                               N  N  R  R  R



PRR-CRB
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X 15 16 17 18 19
pipe:                                              19 19  4  4  4
sent:                                               N  N  R  R  R
RB:                                                       b  b  b



PRR-SSRB
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X 15 16 17 18 19
pipe:                                              19 19  4  5  6
sent:                                               N  N 2R 2R 2R
RB:                                                      bd  d  d
```

Figure 5

In this specific situation, RFC 6675 is more aggressive because once Fast Retransmit is triggered (on the ACK for segment 17), TCP immediately retransmits sufficient data to bring pipe up to cwnd. Our earlier measurements [RFC 6937 section 6] indicates that RFC 6675 significantly outperforms Rate-Halving, PRR-CRB, and some other similarly conservative algorithms that we tested, showing that it is significantly common for the actual losses to exceed the window reduction determined by the congestion control algorithm.

The Linux implementation of Rate-Halving included an early version of the Conservative Reduction Bound [RHweb]. With this algorithm, the 5 ACKs trigger exactly 1 transmission each (2 new data, 3 old data), and cwnd is set to 5. At a window size of 5, it takes 3 round trips

to retransmit all 15 lost segments. Rate-Halving does not raise the window at all during recovery, so when recovery finally completes, TCP will slow start cwnd from 5 up to 10. In this example, TCP operates at half of the window chosen by the congestion control for more than 3 RTTs, increasing the elapsed time and exposing it to timeouts in the event that there are additional losses.

PRR-CRB implements a Conservative Reduction Bound. Since the total losses bring pipe below ssthresh, data is sent such that the total data transmitted, prr_out, follows the total data delivered to the receiver as reported by returning ACKs. Transmission is controlled by the sending limit, which is set to prr_delivered - prr_out. This is indicated by the RB:b tagging in the figure. In this case, PRR-CRB is exposed to exactly the same problems as Rate-Halving; the excess window reduction causes it to take excessively long to recover the losses and exposes it to additional timeouts.

PRR-SSRB increases the window by exactly 1 segment per ACK until pipe rises to ssthresh during recovery. This is accomplished by setting limit to one greater than the data reported to have been delivered to the receiver on this ACK, implementing slow start during recovery, and indicated by RB:d tagging in the figure. Although increasing the window during recovery seems to be ill advised, it is important to remember that this is actually less aggressive than permitted by RFC 5681, which sends the same quantity of additional data as a single burst in response to the ACK that triggered Fast Retransmit.

For less extreme events, where the total losses are smaller than the difference between FlightSize and ssthresh, PRR-CRB and PRR-SSRB have identical behaviors.

8. Properties

The following properties are common to both PRR-CRB and PRR-SSRB, except as noted:

PRR maintains TCP’s ACK clocking across most recovery events, including burst losses. RFC 6675 can send large unclocked bursts following burst losses.

Normally, PRR will spread voluntary window reductions out evenly across a full RTT. This has the potential to generally reduce the burstiness of Internet traffic, and could be considered to be a type of soft pacing. Hypothetically, any pacing increases the probability that different flows are interleaved, reducing the opportunity for ACK compression and other phenomena that increase traffic burstiness. However, these effects have not been quantified.

If there are minimal losses, PRR will converge to exactly the target window chosen by the congestion control algorithm. Note that as TCP approaches the end of recovery, prr_delivered will approach RecoverFS and sndcnt will be computed such that prr_out approaches ssthresh.

Implicit window reductions, due to multiple isolated losses during recovery, cause later voluntary reductions to be skipped. For small numbers of losses, the window size ends at exactly the window chosen by the congestion control algorithm.

For burst losses, earlier voluntary window reductions can be undone by sending extra segments in response to ACKs arriving later during recovery. Note that as long as some voluntary window reductions are not undone, the final value for pipe will be the same as ssthresh, the target cwnd value chosen by the congestion control algorithm.

PRR with either Reduction Bound improves the situation when there are application stalls, e.g., when the sending application does not queue data for transmission quickly enough or the receiver stops advancing rwnd (receiver window). When there is an application stall early during recovery, prr_out will fall behind the sum of the transmissions permitted by sndcnt. The missed opportunities to send due to stalls are treated like banked voluntary window reductions; specifically, they cause prr_delivered - prr_out to be significantly positive. If the application catches up while TCP is still in recovery, TCP will send a partial window burst to catch up to exactly where it would have been had the application never stalled. Although this burst might be viewed as being hard on the network, this is exactly what happens every time there is a partial RTT application stall while not in recovery. We have made the partial RTT stall behavior uniform in all states. Changing this behavior is out of scope for this document.

PRR with Reduction Bound is less sensitive to errors in the pipe estimator. While in recovery, pipe is intrinsically an estimator, using incomplete information to estimate if un-SACKed segments are actually lost or merely out of order in the network. Under some conditions, pipe can have significant errors; for example, pipe is underestimated when a burst of reordered data is prematurely assumed to be lost and marked for retransmission. If the transmissions are regulated directly by pipe as they are with RFC 6675, a step discontinuity in the pipe estimator causes a burst of data, which cannot be retracted once the pipe estimator is corrected a few ACKs later. For PRR, pipe merely determines which algorithm, PRR or the Reduction Bound, is used to compute sndcnt from DeliveredData. While pipe is underestimated, the algorithms are different by at most 1 segment per ACK. Once pipe is updated, they converge to the same final window at the end of recovery.

Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck with no cross traffic, the queue will maintain exactly constant length for the duration of the recovery, except for +1/-1 fluctuation due to differences in packet arrival and exit times. See Appendix A for a detailed discussion of this property.

Although the Strong Packet Conservation Bound is very appealing for a number of reasons, our earlier measurements [RFC 6937 section 6] demonstrate that it is less aggressive and does not perform as well as RFC 6675, which permits bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits TCP to send 1 extra segment per ACK as compared to the Packet Conserving Bound. From the perspective of a strict Packet Conserving Bound, PRR-SSRB does indeed open the window during recovery; however, it is significantly less aggressive than RFC 6675 in the presence of burst losses.

9. Adapting PRR to other transport protocols

The main PRR algorithm and reductions bounds can be adapted to any transport that can support RFC 6675.

The safeACK heuristic can be generalized as any ACK of a retransmission that does not cause some other segment to be marked for retransmission. That is, PRR_SSRB is safe on any ACK that reduces the total number of pending and outstanding retransmissions.

10. Acknowledgements

This document is based in part on previous incomplete work by Matt Mathis, Jeff Semke, and Jamshid Mahdavi [RHID] and influenced by several discussions with John Heffner.

Monia Ghobadi and Sivasankar Radhakrishnan helped analyze the experiments.

Ilpo Jarvinen reviewed the code.

Mark Allman improved the document through his insightful review.

Neal Cardwell for reviewing and testing the patch.

11. Security Considerations

PRR does not change the risk profile for TCP.

Implementers that change PRR from counting bytes to segments have to be cautious about the effects of ACK splitting attacks [Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

12. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

13. Informative References

[CUBIC] Rhee, I. and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant", PFLDnet 2005, February 2005.

[FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM SIGCOMM96, August 1996.

[Flach2016policing]
Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Al Karim, T., Katz-Bassett, E., and R. Govindan, "An Internet-Wide Analysis of Traffic Policing", ACM SIGCOMM SIGCOMM2016, August 2016.

[IMC11] Dukkipati, N., Mathis, M., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany, November 2011.

[Jacobson88]
Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Comput. Commun. Rev. 18(4), August 1988.

[Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", Work in Progress, 16 July 2012.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <https://www.rfc-editor.org/info/rfc3042>.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, DOI 10.17487/RFC3517, April 2003, <https://www.rfc-editor.org/info/rfc3517>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/info/rfc6937>.

[RHID] Mathis, M., Semke, J., and J. Mahdavi, "The Rate-Halving Algorithm for TCP Congestion Control", Work in Progress, August 1999.

[RHweb] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Web publication, December 1997, <http://www.psc.edu/networking/papers/FACKnotes/current/>.

[Savage99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

Appendix A. Strong Packet Conservation Bound

PRR-CRB is based on a conservative, philosophically pure, and aesthetically appealing Strong Packet Conservation Bound, described here. Although inspired by the packet conservation principle [Jacobson88], it differs in how it treats segments that are missing and presumed lost. Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. Note that the effects of presumed losses are included in the pipe calculation, but do not affect the outcome of PRR-CRB, once pipe has fallen below ssthresh.

We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck that is carrying no other traffic, the queue will maintain exactly constant length for the entire duration of the recovery, except for +1/-1 fluctuation due to differences in packet arrival and exit times. Any less aggressive algorithm will result in a declining queue at the bottleneck. Any more aggressive algorithm will result in an increasing queue or additional losses if it is a full drop tail queue.

We demonstrate this property with a little thought experiment:

Imagine a network path that has insignificant delays in both directions, except for the processing time and queue at a single bottleneck in the forward path. By insignificant delay, we mean when a packet is "served" at the head of the bottleneck queue, the following events happen in much less than one bottleneck packet time: the packet arrives at the receiver; the receiver sends an ACK that arrives at the sender; the sender processes the ACK and sends some data; the data is queued at the bottleneck.

If sndcnt is set to DeliveredData and nothing else is inhibiting sending data, then clearly the data arriving at the bottleneck queue will exactly replace the data that was served at the head of the queue, so the queue will have a constant length. If queue is drop tail and full, then the queue will stay exactly full. Losses or reordering on the ACK path only cause wider fluctuations in the queue size, but do not raise its peak size, independent of whether the data is in order or out of order (including loss recovery from an earlier RTT). Any more aggressive algorithm that sends additional data will overflow the drop tail queue and cause loss. Any less aggressive algorithm will under-fill the queue. Therefore, setting sndcnt to DeliveredData is the most aggressive algorithm that does not cause forced losses in this simple network. Relaxing the assumptions

(e.g., making delays more authentic and adding more flows, delayed ACKs, etc.) is likely to increase the fine grained fluctuations in queue size but does not change its basic behavior.

Note that the congestion control algorithm implements a broader notion of optimal that includes appropriately sharing the network. Typical congestion control algorithms are likely to reduce the data sent relative to the Packet Conserving Bound implemented by PRR, bringing TCP's actual window down to ssthresh.

Authors' Addresses

Matt Mathis
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: mattmathis@google.com

Nandita Dukkipati
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: nanditad@google.com

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: ycheng@google.com

TCP Maintenance Working Group
Internet-Draft
Obsoletes: 6937 (if approved)
Intended status: Standards Track
Expires: 17 September 2024

M. Mathis

N. Dukkipati
Y. Cheng
N. Cardwell
Google, Inc.
16 March 2024

Proportional Rate Reduction for TCP
draft-ietf-tcpm-prr-rfc6937bis-08

Abstract

This document updates the experimental Proportional Rate Reduction (PRR) algorithm, described RFC 6937, to standards track. PRR provides logic to regulate the amount of data sent by TCP or other transport protocols during fast recovery. PRR accurately regulates the actual flight size through recovery such that at the end of recovery it will be as close as possible to the slow start threshold (ssthresh), as determined by the congestion control algorithm.

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 https://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 17 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 2
1.1. Document and WG Information 3
2. Background 4
3. Changes From RFC 6937 6
4. Relationships to other standards 7
5. Definitions 8
6. Algorithm 9
7. Examples 10
8. Properties 13
9. Adapting PRR to other transport protocols 15
10. Acknowledgements 15
11. IANA Considerations 15
12. Security Considerations 15
13. Normative References 15
14. Informative References 16
Appendix A. Strong Packet Conservation Bound 17
Authors' Addresses 18

1. Introduction

This document updates the Proportional Rate Reduction (PRR) algorithm described in [RFC6937] from experimental to standards track. PRR smoothly regulates the amount of data sent during fast recovery, such that at the end of recovery the flight size will be as close as possible to the slow start threshold (ssthresh), as determined by the congestion control algorithm. PRR has been deployed in at least three major TCP implementations covering the vast majority of today's web traffic.

This document specifies several main changes from RFC 6937. First, it introduces a new heuristic that replaces a manual configuration parameter that determined how conservative PRR was when the "pipe" of in-flight data was less than ssthresh. Second, the algorithm specifies behavior for non-SACK connections. Third, the algorithm ensures a smooth sending process even when the sender has experienced high reordering and starts loss recovery after a large amount of sequence space has been SACKed. Finally, this document also includes additional discussion about the integration of PRR with congestion control and lost detection algorithms.

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].

1.1. Document and WG Information

Formatted: 2024-03-16 15:20:05+00:00

Please send all comments, questions and feedback to tcpm@ietf.org

About revision 00:

The introduction above was drawn from draft-mathis-tcpm-rfc6937bis-00. All of the text below was copied verbatim from RFC 6937, to facilitate comparison between RFC 6937 and this document as it evolves.

About revision 01:

* Recast the RFC 6937 introduction as background

* Made "Changes From RFC 6937" an explicit section

* Made Relationships to other standards more explicit

* Added a generalized safeACK heuristic

* Provided hints for non TCP implementations

* Added language about detecting ACK splitting, but have no advice on actions (yet)

About revision 02:

* Companion RACK loss detection RECOMMENDED

* Non-SACK accounting in the pseudo code

* cwnd computation in the pseudo code

* Force fast retransmit at the beginning of fast recovery

* Remove deprecated Rate-Halving text

* Fixed bugs in the example traces

About revision 03 and 04:

* Clarify when and how sndcnt becomes 0

* Improve algorithm to smooth the sending rate under higher reordering cases

About revision 05:

* Revert the RecoverFS text and pseudocode to match the behavior in draft revision 03 and more closely match Linux TCP PRR

About revision 06:

* Update RecoverFS to be initialized as: RecoverFS = pipe.

About revision 07:

* Restored the revision 04 prose description for the rationale for initializing RecoverFS as: RecoverFS = pipe.

* Added reference to [Hoe96Startup] in acknowledgements

About revision 08:

* Inserted missing reference to [RFC9293]

* Recategorized "voluntary window reductions" as a phrase introduced by PRR

2. Background

Standard congestion control [RFC5681] requires that TCP (and other protocols) reduce their congestion window (cwnd) in response to losses. Fast recovery, described in the same document, is the reference algorithm for making this adjustment. Its stated goal is to recover TCP's self clock by relying on returning ACKs during recovery to clock more data into the network. Fast recovery typically adjusts the window by waiting for one half round-trip time (RTT) of ACKs to pass before sending any data. It is fragile because it cannot compensate for the implicit window reduction caused by the losses themselves.

[RFC6675] makes fast recovery with Selective Acknowledgement (SACK) [RFC2018] more accurate by computing "pipe", a sender side estimate of the number of bytes still outstanding in the network. With [RFC6675], fast recovery is implemented by sending data as necessary on each ACK to prevent pipe from falling below ssthresh, the window size as determined by the congestion control algorithm. This protects fast recovery from timeouts in many cases where there are

heavy losses, although not if the entire second half of the window of data or ACKs are lost. However, a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a burst of data.

PRR avoids these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size as determined by the congestion control algorithm. It uses the fraction that is appropriate for the target window chosen by the congestion control algorithm. During PRR, one of two additional Reduction Bound algorithms limits the total window reduction due to all mechanisms, including transient application stalls and the losses themselves.

We describe two slightly different Reduction Bound algorithms: Conservative Reduction Bound (CRB), which is strictly packet conserving; and a Slow Start Reduction Bound (SSRB), which is more aggressive than CRB by, at most, 1 segment per ACK. PRR-CRB meets the Strong Packet Conservation Bound described in Appendix A; however, in real networks it does not perform as well as the algorithms described in [RFC6675], which prove to be more aggressive in a significant number of cases. SSRB offers a compromise by allowing TCP to send 1 additional segment per ACK relative to CRB in some situations. Although SSRB is less aggressive than [RFC6675] (transmitting fewer segments or taking more time to transmit them), it outperforms due to the lower probability of additional losses during recovery.

The Strong Packet Conservation Bound on which PRR and both Reduction Bounds are based is patterned after Van Jacobson's packet conservation principle: segments delivered to the receiver are used as the clock to trigger sending the same number of segments back into the network. As much as possible, PRR and the Reduction Bound algorithms rely on this self clock process, and are only slightly affected by the accuracy of other estimators, such as pipe [RFC6675] and cwnd. This is what gives the algorithms their precision in the presence of events that cause uncertainty in other estimators.

The original definition of the packet conservation principle [Jacobson88] treated packets that are presumed to be lost (e.g., marked as candidates for retransmission) as having left the network. This idea is reflected in the pipe estimator defined in [RFC6675] and used here, but it is distinct from the Strong Packet Conservation Bound as described in Appendix A, which is defined solely on the basis of data arriving at the receiver.

3. Changes From RFC 6937

The largest change since [RFC6937] is the introduction of a new heuristic that uses good recovery progress (for TCP, when the latest ACK advances snd.una and does not indicate that a prior fast retransmit has been lost) to select the Reduction Bound. [RFC6937] left the choice of Reduction Bound to the discretion of the implementer but recommended to use SSRB by default. For all of the environments explored in earlier PRR research, the new heuristic is consistent with the old recommendation.

The paper "An Internet-Wide Analysis of Traffic Policing" [Flach2016policing] uncovered a crucial situation not previously explored, where both Reduction Bounds perform very poorly, but for different reasons. Under many configurations, token bucket traffic policers can suddenly start discarding a large fraction of the traffic when tokens are depleted, without any warning to the end systems. The transport congestion control has no opportunity to measure the token rate, and sets ssthresh based on the previously observed path performance. This value for ssthresh may cause a data rate that is substantially larger than the token replenishment rate, causing high loss. Under these conditions, both reduction bounds perform very poorly. PRR-CRB is too timid, sometimes causing very long recovery times at smaller than necessary windows, and PRR-SSRB is too aggressive, often causing many retransmissions to be lost for multiple rounds. Both cases lead to prolonged recovery, decimating application latency and/or goodput.

Investigating these environments led to the development of a "safeACK" heuristic to dynamically switch between Reduction Bounds: by default conservatively use PRR-CRB and only switch to PRR-SSRB when ACKs indicate the recovery is making good progress (snd.una is advancing without detecting any new losses). The SafeACK heuristic was experimented with in Google’s CDN [Flach2016policing] and implemented in Linux since 2015.

This heuristic is only invoked where losses, application-limited behavior, or other events cause the current "pipe" or flight size to fall below ssthresh. The high loss rates that make the heuristic essential are only common in the presence of heavy losses such as traffic policers [Flach2016policing]. In these environments the heuristic serves to salvage a bad situation and any reasonable implementation of the heuristic performs far better than either bound by itself.

Another PRR algorithm change improves the sending process when the sender enters recovery after a large portion of sequence space has been SACKed. This scenario could happen when the sender has

previously detected reordering, for example, by using [RFC8985]. In the previous version of PRR, RecoverFS did not properly account for sequence ranges SACKed before entering fast recovery, which caused PRR to send too slow initially. With the change, PRR initializes RecoverFS to the pipe variable for SACK connections.

Yet another change is to force a fast retransmit upon the first ACK that triggers the recovery. Previously, PRR may not allow a fast retransmit (i.e. sndcnt is 0) on the first ACK in fast recovery, depending on the loss situation. Forcing a fast retransmit is important to maintain the ACK clock and avoid potential RTO events. The forced fast retransmit only happens once during the entire recovery and still follows the packet conservation principles in PRR. This heuristic has been implemented since the first widely deployed TCP PRR implementation in 2011.

Since [RFC6937] was written, PRR has also been adapted to perform multiplicative window reduction for non-loss based congestion control algorithms, such as for [RFC3168] style ECN. This is typically done by using some parts of the loss recovery state machine (in particular the RecoveryPoint from [RFC6675]) to invoke the PRR ACK processing for exactly one round trip worth of ACKs.

For [RFC6937] we published a companion paper [IMC11] in which we evaluated [RFC3517] and various experimental PRR versions in a large scale measurement study. Today, the legacy algorithms used in that study have already faded from code bases, making such comparisons impossible without recreating historical algorithms. Readers interested in the measurement study should review section 5 of RFC 6937 and the IMC paper [IMC11].

4. Relationships to other standards

PRR is described as modifications to "TCP Congestion Control" [RFC5681], and "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP" [RFC6675]. It is most accurate with SACK [RFC2018] but does not require SACK.

The SafeACK heuristic came about as a result of robust Lost Retransmission Detection under development in an early precursor to [RFC8985]. Without LRD, policers that cause very high loss rates are at very high risk of causing retransmission timeouts because both [RFC5681] and [RFC6675] will send retransmissions above the policed rate. It is RECOMMENDED that PRR is implemented together with RACK-TLP loss recovery [RFC8985].

5. Definitions

The following terms, parameters, and state variables are used as they are defined in earlier documents:

[RFC9293]: snd.una (send unacknowledged).

[RFC5681]: duplicate ACK, FlightSize, Sender Maximum Segment Size (SMSS).

[RFC6675]: pipe, covered (as in "covered sequence numbers").

PRR defines additional variables and terms:

DeliveredData: The number of bytes newly delivered by the most recent ACK. With SACK, DeliveredData is the change in snd.una plus the bytes newly selectively acknowledged. Without SACK, DeliveredData is the change in snd.una for a partial ACK or 1 MSS worth of bytes for a DUPACK. Note that without SACK, a poorly-behaved receiver that returns extraneous DUPACKs as described in [Savage99] can artificially inflate DeliveredData. As a mitigation, PRR disallows incrementing DeliveredData when the total bytes delivered exceeds the outstanding data upon entering recovery (i.e., RecoverFS).

RecoverFS: The "recovery flight size", the number of bytes the sender estimates are in flight in the network upon entering fast recovery. PRR uses RecoverFS to compute a smooth sending rate. Upon entering fast recovery, PRR initializes RecoverFS to the sender’s best estimate of the number of bytes outstanding in the network; for connections with SACK this is typically "pipe" as specified in RFC 6675. RecoverFS remains constant during a given fast recovery episode.

safeACK: A local boolean variable indicating that the current ACK reported good progress. SafeACK is true only when the ACK has cumulatively acknowledged new data and the ACK does not indicate further losses. For example, an ACK triggering RFC6675 "last resort" retransmission (Section 4, NextSeg() condition 4) may indicate further losses. Both conditions indicate the recovery is making good progress and can send more aggressively.

sndcnt: A local variable indicating exactly how many bytes should be sent in response to each ACK. Note that the decision of which data to send (e.g., retransmit missing data or send more new data) is out of scope for this document.

Voluntary window reductions: choosing not to send data in response to some ACKs, for the purpose of reducing the sending window size and data rate.

6. Algorithm

At the beginning of recovery, initialize the PRR state. This assumes a modern congestion control algorithm, CongCtrlAlg(), that might set ssthresh to something other than FlightSize/2:

```
ssthresh = CongCtrlAlg()      // Target flight size in recovery
prr_delivered = 0             // Total bytes delivered in recovery
prr_out = 0                   // Total bytes sent in recovery
pipe = (RFC 6675 pipe algorithm)
RecoverFS = pipe              // RFC 6675 pipe before recovery
```

```
On every ACK starting or during fast recovery:
   DeliveredData = (bytes newly cumulatively acknowledged)
   if (SACK is used) {
      DeliveredData += (bytes newly selectively acknowledged)
   } else if (ACK is a DUPACK and prr_delivered < RecoverFS) {
      DeliveredData += MSS
   }
   if (DeliveredData is 0)
      Return


   prr_delivered += DeliveredData
   pipe = (RFC 6675 pipe algorithm)
   safeACK = (snd.una advances and no further loss indicated)
   if (pipe > ssthresh) {
      // Proportional Rate Reduction
      sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
   } else {
      // PRR-CRB by default
      sndcnt = MAX(prr_delivered - prr_out, DeliveredData)
      if (safeACK) {
         // PRR-SSRB when recovery is in good progress
         sndcnt += MSS
      }
      // Attempt to catch up, as permitted
      sndcnt = MIN(ssthresh - pipe, sndcnt)
   }


   if (prr_out is 0 AND sndcnt is 0) {
      // Force a fast retransmit upon entering recovery
      sndcnt = MSS
   }
   cwnd = pipe + sndcnt

On any data transmission or retransmission:
   prr_out += (data sent)
```

7. Examples

We illustrate these algorithms by showing their different behaviors for two scenarios: TCP experiencing either a single loss or a burst of 15 consecutive losses. In all cases we assume bulk data (no application pauses), standard Additive Increase Multiplicative Decrease (AIMD) congestion control [RFC5681], and cwnd = FlightSize = pipe = 20 segments, so ssthresh will be set to 10 at the beginning of recovery. We also assume standard Fast Retransmit and Limited Transmit [RFC3042], so TCP will send 2 new segments followed by 1

retransmit in response to the first 3 duplicate ACKs following the losses.

Each of the diagrams below shows the per ACK response to the first round trip for the various recovery algorithms when the zeroth segment is lost. The top line indicates the transmitted segment number triggering the ACKs, with an X for the lost segment. "cwnd" and "pipe" indicate the values of these algorithms after processing each returning ACK but before further (re)transmission. "Sent" indicates how much 'N'ew or 'R'etransmitted data would be sent. Note that the algorithms for deciding which data to send are out of scope of this document.

```
RFC 6675
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
pipe:    19 19 18 18 17 16 15 14 13 12 11 10 10 10 10 10 10 10 10
sent:     N  N  R                         N  N  N  N  N  N  N  N



PRR
ack#   X  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
cwnd:    20 20 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 10
pipe:    19 19 18 18 17 17 16 16 15 15 14 14 13 13 12 12 11 11 10
sent:     N  N  R     N     N     N     N     N     N     N     N
```

Figure 1

Note that both algorithms send the same total amount of data. RFC 6675 experiences a "half window of silence" while PRR spreads the voluntary window reduction across an entire RTT.

Next, we consider the same initial conditions when the first 15 packets (0-14) are lost. During the remainder of the lossy RTT, only 5 ACKs are returned to the sender. We examine each of these algorithms in succession.

```
RFC 6675
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  15 16 17 18 19
cwnd:                                               20 20 11 11 11
pipe:                                               19 19  4 10 10
sent:                                                N  N 7R  R  R



PRR
ack#   X  X  X  X  X  X  X  X  X  X  X  X  X  X  X  15 16 17 18 19
cwnd:                                               20 20  5  5  5
pipe:                                               19 19  4  4  4
sent:                                                N  N  R  R  R
```

Figure 2

In this specific situation, RFC 6675 is more aggressive because once Fast Retransmit is triggered (on the ACK for segment 17), TCP immediately retransmits sufficient data to bring pipe up to cwnd. Our earlier measurements [RFC 6937 section 6] indicates that RFC 6675 significantly outperforms PRR, and some other similarly conservative algorithms that we tested, showing that it is significantly common for the actual losses to exceed the window reduction determined by the congestion control algorithm.

Under such heavy losses, PRR uses the PRR-CRB to follow the packet conservation principle. Since the total losses bring pipe below ssthresh, data is sent such that the total data transmitted, prr_out, follows the total data delivered to the receiver as reported by returning ACKs. Transmission is controlled by the sending limit, which is set to prr_delivered - prr_out. PRR-CRB conservative window reduction causes it to take excessively long to recover the losses and exposes it to additional timeouts.

While not shown in the figure above, once the fast retransmits sent upon ACK#17 deliver and solicit further ACKs that increment the snd.una, PRR enters PRR-SSRB and increases the window by exactly 1 segment per ACK until pipe rises to ssthresh during recovery. On heavy losses when cwnd is large, PRR-SSRB recovers the losses exponentially faster than PRR-CRB. Although increasing the window during recovery seems to be ill advised, it is important to remember that this is actually less aggressive than permitted by RFC 5681, which sends the same quantity of additional data as a single burst in response to the ACK that triggered Fast Retransmit.

For less severe loss events, where the total losses are smaller than the difference between FlightSize and ssthresh, PRR-CRB and PRR-SSRB are not invoked since PRR stays in the proportional rate reduction mode.

8. Properties

The following properties are common to both PRR-CRB and PRR-SSRB, except as noted:

PRR maintains TCP's ACK clocking across most recovery events, including burst losses. RFC 6675 can send large unclocked bursts following burst losses.

Normally, PRR will spread voluntary window reductions out evenly across a full RTT. This has the potential to generally reduce the burstiness of Internet traffic, and could be considered to be a type of soft pacing. Hypothetically, any pacing increases the probability that different flows are interleaved, reducing the opportunity for ACK compression and other phenomena that increase traffic burstiness. However, these effects have not been quantified.

If there are minimal losses, PRR will converge to exactly the target window chosen by the congestion control algorithm. Note that as TCP approaches the end of recovery, prr_delivered will approach RecoverFS and sndcnt will be computed such that prr_out approaches ssthresh.

Implicit window reductions, due to multiple isolated losses during recovery, cause later voluntary reductions to be skipped. For small numbers of losses, the window size ends at exactly the window chosen by the congestion control algorithm.

For burst losses, earlier voluntary window reductions can be undone by sending extra segments in response to ACKs arriving later during recovery. Note that as long as some voluntary window reductions are not undone, and there is no application stall, the final value for pipe will be the same as ssthresh, the target cwnd value chosen by the congestion control algorithm.

PRR with either Reduction Bound improves the situation when there are application stalls, e.g., when the sending application does not queue data for transmission quickly enough or the receiver stops advancing rwnd (receiver window). When there is an application stall early during recovery, prr_out will fall behind the sum of transmissions allowed by sndcnt. The missed opportunities to send due to stalls are treated like banked voluntary window reductions; specifically, they cause prr_delivered - prr_out to be significantly positive. If the application catches up while TCP is still in recovery, TCP will

send a partial window burst to catch up to exactly where it would have been had the application never stalled. Although this burst might be viewed as being hard on the network, this is exactly what happens every time there is a partial RTT application stall while not in recovery. We have made partial RTT stall behavior uniform in all states. Changing this behavior is out of scope for this document.

PRR with Reduction Bound is less sensitive to errors in the pipe estimator. While in recovery, pipe is intrinsically an estimator, using incomplete information to estimate if un-SACKed segments are actually lost or merely out of order in the network. Under some conditions, pipe can have significant errors; for example, pipe is underestimated when a burst of reordered data is prematurely assumed to be lost and marked for retransmission. If the transmissions are regulated directly by pipe as they are with RFC 6675, a step discontinuity in the pipe estimator causes a burst of data, which cannot be retracted once the pipe estimator is corrected a few ACKs later. For PRR, pipe merely determines which algorithm, PRR or the Reduction Bound, is used to compute sndcnt from DeliveredData. While pipe is underestimated, the algorithms are different by at most 1 segment per ACK. Once pipe is updated, they converge to the same final window at the end of recovery.

Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck with no cross traffic, the queue will maintain exactly constant length for the duration of the recovery, except for +1/-1 fluctuation due to differences in packet arrival and exit times. See Appendix A for a detailed discussion of this property.

Although the Strong Packet Conservation Bound is very appealing for a number of reasons, our earlier measurements [RFC 6937 section 6] demonstrate that it is less aggressive and does not perform as well as RFC 6675, which permits bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits TCP to send 1 extra segment per ACK as compared to the Packet Conserving Bound when the ACK indicates the recovery is in good progress without further losses. From the perspective of a strict Packet Conserving Bound, PRR-SSRB does indeed open the window during recovery; however, it is significantly less aggressive than RFC 6675 in the presence of burst losses. The RFC 6675 "half window of silence" may temporarily reduce queue pressure when congestion control does not reduce the congestion window entering recovery to avoid further losses. The goal of PRR is to minimize the opportunities to lose the self clock by accurately

controlling flightsize to the target set by the congestion control. It is the congestion control's responsibility to avoid a full queue, not PRR.

9. Adapting PRR to other transport protocols

The main PRR algorithm and reductions bounds can be adapted to any transport that can support RFC 6675. In one major implementation (Linux TCP), PRR has been the default fast recovery algorithm for its default and supported congestion control modules.

The safeACK heuristic can be generalized as any ACK of a retransmission that does not cause some other segment to be marked for retransmission. That is, PRR_SSRB is safe on any ACK that reduces the total number of pending and outstanding retransmissions.

10. Acknowledgements

This document is based in part on previous work by Janey C. Hoe (see section 3.2, "Recovery from Multiple Packet Losses", of [Hoe96Startup]) and Matt Mathis, Jeff Semke, and Jamshid Mahdavi [RHID], and influenced by several discussions with John Heffner.

Monia Ghobadi and Sivasankar Radhakrishnan helped analyze the experiments. Ilpo Jarvinen reviewed the initial implementation. Mark Allman, Richard Scheffenegger, and Markku Kojo improved the document through their insightful reviews.

11. IANA Considerations

This memo includes no request to IANA.

12. Security Considerations

PRR does not change the risk profile for TCP.

Implementers that change PRR from counting bytes to segments have to be cautious about the effects of ACK splitting attacks [Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

13. Normative References

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/info/rfc8985>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

14. Informative References

[Flach2016policing]
Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Al Karim, T., Katz-Bassett, E., and R. Govindan, "An Internet-Wide Analysis of Traffic Policing", ACM SIGCOMM SIGCOMM2016, August 2016.

[Hoe96Startup]
Hoe, J., "Improving the start-up behavior of a congestion control scheme for TCP", ACM SIGCOMM SIGCOMM1996, August 1996.

[IMC11] Dukkipati, N., Mathis, M., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany, November 2011.

[Jacobson88]
Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Comput. Commun. Rev. 18(4), August 1988.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <https://www.rfc-editor.org/info/rfc3042>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, DOI 10.17487/RFC3517, April 2003, <https://www.rfc-editor.org/info/rfc3517>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/info/rfc6937>.

[RHID] Mathis, M., Semke, J., and J. Mahdavi, "The Rate-Halving Algorithm for TCP Congestion Control", Work in Progress, August 1999, <https://datatracker.ietf.org/doc/html/draft-mathis-tcp-ratehalving>.

[Savage99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

Appendix A. Strong Packet Conservation Bound

PRR-CRB is based on a conservative, philosophically pure, and aesthetically appealing Strong Packet Conservation Bound, described here. Although inspired by the packet conservation principle [Jacobson88], it differs in how it treats segments that are missing and presumed lost. Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. Note that the effects of presumed losses are included in the pipe calculation, but do not affect the outcome of PRR-CRB, once pipe has fallen below ssthresh.

We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck that is carrying no other traffic, the queue will maintain exactly constant length for the entire duration of the recovery, except for +1/-1 fluctuation due to differences in

packet arrival and exit times. Any less aggressive algorithm will result in a declining queue at the bottleneck. Any more aggressive algorithm will result in an increasing queue or additional losses if it is a full drop tail queue.

We demonstrate this property with a little thought experiment:

Imagine a network path that has insignificant delays in both directions, except for the processing time and queue at a single bottleneck in the forward path. By insignificant delay, we mean when a packet is "served" at the head of the bottleneck queue, the following events happen in much less than one bottleneck packet time: the packet arrives at the receiver; the receiver sends an ACK that arrives at the sender; the sender processes the ACK and sends some data; the data is queued at the bottleneck.

If sndcnt is set to DeliveredData and nothing else is inhibiting sending data, then clearly the data arriving at the bottleneck queue will exactly replace the data that was served at the head of the queue, so the queue will have a constant length. If queue is drop tail and full, then the queue will stay exactly full. Losses or reordering on the ACK path only cause wider fluctuations in the queue size, but do not raise its peak size, independent of whether the data is in order or out of order (including loss recovery from an earlier RTT). Any more aggressive algorithm that sends additional data will overflow the drop tail queue and cause loss. Any less aggressive algorithm will under-fill the queue. Therefore, setting sndcnt to DeliveredData is the most aggressive algorithm that does not cause forced losses in this simple network. Relaxing the assumptions (e.g., making delays more authentic and adding more flows, delayed ACKs, etc.) is likely to increase the fine grained fluctuations in queue size but does not change its basic behavior.

Note that the congestion control algorithm implements a broader notion of optimal that includes appropriately sharing the network. Typical congestion control algorithms are likely to reduce the data sent relative to the Packet Conserving Bound implemented by PRR, bringing TCP’s actual window down to ssthresh.

Authors’ Addresses

Matt Mathis
Email: ietf@mattmathis.net

Nandita Dukkipati
Google, Inc.
Email: nanditad@google.com

Yuchung Cheng
Google, Inc.
Email: ycheng@google.com

Neal Cardwell
Google, Inc.
Email: ncardwell@google.com

TCPM
Internet-Draft
Obsoletes: 8312 (if approved)
Updates: 5681 (if approved)
Intended status: Standards Track
Expires: 28 April 2022

L. Xu
UNL
S. Ha
Colorado
I. Rhee
Bowery
V. Goel
Apple Inc.
L. Eggert, Ed.
NetApp
25 October 2021

CUBIC for Fast and Long-Distance Networks
draft-ietf-tcpm-rfc8312bis-05

Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of the linear window increase function on the sender side to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting RFC 8312. This also requires updating RFC 5681, to allow for CUBIC's occasionally more aggressive sending behavior.

Note to Readers

Discussion of this draft takes place on the TCPM working group mailing list (mailto:tcpm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tcpm/.

Working Group information can be found at https://datatracker.ietf.org/wg/tcpm/; source code and issues list for this draft can be found at https://github.com/NTAP/rfc8312bis.

Note to the RFC Editor

xml2rfc currently renders in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/596)

Also, please manually change "Figure" to "Equation" for all artwork with anchors beginning with "eq" - xml2rfc doesn't seem to be able to do this.

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 https://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 28 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 4
2. Conventions 5
3. Design Principles of CUBIC 5
  3.1. Principle 1 for the CUBIC Increase Function 5
  3.2. Principle 2 for Reno-Friendliness 6
  3.3. Principle 3 for RTT Fairness 7
  3.4. Principle 4 for the CUBIC Decrease Factor 7
4. CUBIC Congestion Control 8

4.1. Definitions 8
4.1.1. Constants of Interest 8
4.1.2. Variables of Interest 8
4.2. Window Increase Function 9
4.3. Reno-Friendly Region 11
4.4. Concave Region 13
4.5. Convex Region 13
4.6. Multiplicative Decrease 14
4.7. Fast Convergence 15
4.8. Timeout 15
4.9. Spurious Congestion Events 16
4.10. Slow Start 17
5. Discussion 17
5.1. Fairness to Reno 18
5.2. Using Spare Capacity 20
5.3. Difficult Environments 21
5.4. Investigating a Range of Environments 21
5.5. Protection against Congestion Collapse 22
5.6. Fairness within the Alternative Congestion Control Algorithm 22
5.7. Performance with Misbehaving Nodes and Outside Attackers 22
5.8. Behavior for Application-Limited Flows 22
5.9. Responses to Sudden or Transient Events 22
5.10. Incremental Deployment 23
6. Security Considerations 23
7. IANA Considerations 23
8. References 23
8.1. Normative References 23
8.2. Informative References 25
Appendix A. Acknowledgments 27
Appendix B. Evolution of CUBIC 28
B.1. Since draft-ietf-tcpm-rfc8312bis-04 28
B.2. Since draft-ietf-tcpm-rfc8312bis-03 29
B.3. Since draft-ietf-tcpm-rfc8312bis-02 29
B.4. Since draft-ietf-tcpm-rfc8312bis-01 29
B.5. Since draft-ietf-tcpm-rfc8312bis-00 30
B.6. Since draft-eggert-tcpm-rfc8312bis-03 30
B.7. Since draft-eggert-tcpm-rfc8312bis-02 30
B.8. Since draft-eggert-tcpm-rfc8312bis-01 30
B.9. Since draft-eggert-tcpm-rfc8312bis-00 30
B.10. Since RFC8312 31
B.11. Since the Original Paper 31
Authors' Addresses 32

1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over classical Reno congestion control [RFC5681]. It is therefore to be regarded as the currently most widely deployed standard for TCP congestion control. CUBIC can also be used for other transport protocols such as QUIC [RFC9000] and SCTP [RFC4960] as a default congestion controller.

The design of CUBIC was motivated by the well-documented problem classical Reno TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HKLRX06] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC4960], TFRC [RFC5348], and QUIC congestion control [RFC9002], which use the same linear increase function for window growth. We refer to all Reno-style standards and their variants collectively as "Reno" below.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of classical Reno to remedy this problem. Specifically, CUBIC uses a cubic function instead of the linear window increase function of Reno to improve scalability and stability under fast and long-distance networks.

This document updates the specification of CUBIC to include algorithmic improvements based on the Linux, Windows, and Apple implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312]. This requires an update to [RFC5681], which limits the aggressiveness of Reno TCP implementations in its Section 3. Since CUBIC is occasionally more aggressive than the [RFC5681] algorithms, this document updates [RFC5681] to allow for CUBIC’s behavior.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Reno in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2: To be Reno-friendly, CUBIC is designed to behave like Reno in networks with short RTTs and small bandwidth where Reno performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Reno increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event detected by duplicate ACKs, Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], TCP RACK [RFC8985] or QUIC loss detection [RFC9002], CUBIC remembers the congestion window size at which it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX07]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

3.2. Principle 2 for Reno-Friendliness

CUBIC promotes per-flow fairness to Reno. Note that Reno performs well over paths with short RTTs and small bandwidths (or small BDPs). There is only a scalability problem in networks with long RTTs and large bandwidths (or large BDPs).

A congestion control algorithm designed to be friendly to Reno on a per-flow basis must increase its congestion window less aggressively in small BDP networks than in large BDP networks.

The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than Reno, CUBIC simply follows the window size of Reno to ensure that CUBIC achieves at least the same throughput as Reno in small-BDP networks. We call this region where CUBIC behaves like Reno the "Reno-friendly region".

3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the Reno-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the Reno-friendly region.

This notion of a linear throughput ratio is similar to that of Reno under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of Reno flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04].

CUBIC always ensures a linear throughput ratio independent of the amount of statistical multiplexing. This is an improvement over Reno. While there is no consensus on particular throughput ratios for different RTT flows, we believe that over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of Reno under low statistical multiplexing environments).

3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas Reno uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that HighSpeed TCP (HSTCP) [RFC3649] and other approaches (e.g., [GV02]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that in environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

4. CUBIC Congestion Control

In this section, we discuss how the congestion window is updated during the different stages of the CUBIC congestion controller.

4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Implementations can use bytes to express window sizes, which would require factoring in the maximum segment size wherever necessary and replacing _segments_acked_ with the number of bytes acknowledged in Figure 4.

4.1.1. Constants of Interest

__cubic_: CUBIC multiplicative decrease factor as described in Section 4.6.

__cubic_: CUBIC additive increase factor used in Reno-friendly region as described in Section 4.3.

_C_: constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high BDP networks. Please see Section 5 for more explanation on how it is set. The unit for _C_ is

$$\frac{\text{segment}}{\text{second}^3}$$

4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:

RTT: Smoothed round-trip time in seconds, calculated as described in [RFC6298].

cwnd: Current congestion window in segments.

ssthresh: Current slow start threshold in segments.

_W_max_: Size of _cwnd_ in segments just before _cwnd_ was reduced in the last congestion event when fast convergence is disabled. However, if fast convergence is enabled, the size may be further reduced based on the current saturation point.

_K_: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to _W_max_.

_current_time_: Current time of the system in seconds.

_epoch_start_: The time in seconds at which the current congestion avoidance stage started.

_cwnd_start_: The _cwnd_ at the beginning of the current congestion avoidance stage, i.e., at time _epoch_start_.

W_cubic(_t_): The congestion window in segments at time _t_ in seconds based on the cubic increase function, as described in Section 4.2.

target: Target value of congestion window in segments after the next RTT, that is, W_cubic(_t_ + _RTT_), as described in Section 4.2.

_W_est_: An estimate for the congestion window in segments in the Reno-friendly region, that is, an estimate for the congestion window of Reno.

_segments_acked_: Number of MSS-sized segments acked when a "new ACK" is received, i.e., an ACK that cumulatively acknowledges the delivery of new data. This number will be a decimal value when a new ACK acknowledges an amount of data that is not MSS-sized. Specifically, it can be less than 1 when a new ACK acknowledges a segment smaller than the MSS.

4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Reno by increasing the congestion window only at the reception of a new ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].

During congestion avoidance, after a congestion event is detected by mechanisms described in Section 3.1, CUBIC changes the window increase function of Reno.

CUBIC uses the following window increase function:

$$W_{cubic}(t) = C * (t - K)^3 + W_{max}$$

Figure 1

where _t_ is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

$$t = current_time - epoch_{start}$$

and where _epoch_start_ is the time at which the current congestion avoidance stage starts. _K_ is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to _W_max_ if there are no further congestion events and is calculated using the following equation:

$$K = \sqrt[3]{\frac{W_{max} - cwnd_{start}}{C}}$$

Figure 2

where _cwnd_start_ is the congestion window at the beginning of the current congestion avoidance stage.

Upon receiving a new ACK during congestion avoidance, CUBIC computes the _target_ congestion window size after the next _RTT_ using Figure 1 as follows, where _RTT_ is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC's congestion window increase rate is non-decreasing and is less than the increase rate of slow start.

```
                  /
                  |                if W     (t + RTT) < cwnd
                  |cwnd               cubic
                  |
                  |
                  |
        target = <                 if W     (t + RTT) > 1.5 * cwnd
                  |1.5 * cwnd         cubic
                  |
                  |
                  |W     (t + RTT)
                  | cubic          otherwise
                  \
```

The elapsed time _t_ in Figure 1 MUST NOT include periods during which _cwnd_ has not been updated due to an application limit (see Section 5.8).

Depending on the value of the current congestion window size _cwnd_, CUBIC runs in three different regions:

1. The Reno-friendly region, which ensures that CUBIC achieves at least the same throughput as Reno.

2. The concave region, if CUBIC is not in the Reno-friendly region and _cwnd_ is less than _W_max_.

3. The convex region, if CUBIC is not in the Reno-friendly region and _cwnd_ is greater than _W_max_.

Below, we describe the exact actions taken by CUBIC in each region.

4.3. Reno-Friendly Region

Reno performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the Reno-friendly region to achieve at least the same throughput as Reno.

The Reno-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of (segments per _RTT_) and a multiplicative factor of , denoted by AIMD(,). _p_ is the packet loss rate. Specifically, the average congestion window size of AIMD(,) can be calculated using Figure 3.

```
                                  _______________
                                 /   * (1 + )
                AVG_AIMD(, ) = | / ---------------
                               |/  2 * (1 - ) * p
```

Figure 3

By the same analysis, to achieve the same average window size as Reno that uses AIMD(1, 0.5), must be equal to,

```
                                 1 -
                             3 * -----
                                 1 +
```

Thus, CUBIC uses Figure 4 to estimate the window size _W_est_ in the Reno-friendly region with

```
                                   1 -
                                        cubic
                             = 3 * ----------
                       cubic       1 +
                                        cubic
```

which achieves the same average window size as Reno. When receiving a new ACK in congestion avoidance (where _cwnd_ could be greater than or less than _W_max_), CUBIC checks whether W_cubic(_t_) is less than _W_est_. If so, CUBIC is in the Reno-friendly region and _cwnd_ SHOULD be set to _W_est_ at each reception of a new ACK.

_W_est_ is set equal to _cwnd_start_ at the start of the congestion avoidance stage. After that, on every new ACK, _W_est_ is updated using Figure 4. Note that this equation is for a connection where Appropriate Byte Counting (ABC) [RFC3465] is disabled. For a connection with ABC enabled, this equation SHOULD be adjusted by using the number of acknowledged bytes instead of acknowledged segments. Also note that this equation works for connections with enabled or disabled Delayed ACKs [RFC5681], as _segments_acked_ will be different based on the segments actually acknowledged by a new ACK.

```
                                     segments_acked
                W    = W    +       * --------------
                 est    est    cubic        cwnd
```

Figure 4

Note that once _W_est_ reaches _W_max_, that is, _W_est_ >= _W_max_, CUBIC needs to start probing to determine the new value of _W_max_. At this point, __cubic_ SHOULD be set to 1 to ensure that CUBIC can achieve the same congestion window increment as Reno, which uses AIMD(1, 0.5).

4.4. Concave Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and _cwnd_ is less than _W_max_, then CUBIC is in the concave region. In this region, _cwnd_ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where _target_ is calculated as described in Section 4.2.

4.5. Convex Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and _cwnd_ is larger than or equal to _W_max_, then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

Unless it is overridden by the AIMD window increase, CUBIC is very careful in this region. The convex profile aims to increase the window very slowly at the beginning when _cwnd_ is around _W_max_ and then gradually increases its rate of increase. We also call this region the "maximum probing phase", since CUBIC is searching for a new _W_max_. In this region, _cwnd_ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where _target_ is calculated as described in Section 4.2.

4.6. Multiplicative Decrease

When a congestion event is detected by mechanisms described in Section 3.1, CUBIC updates _W_max_ and reduces _cwnd_ and _ssthresh_ immediately as described below. In case of packet loss, the sender MUST reduce _cwnd_ and _ssthresh_ immediately upon entering loss recovery, similar to [RFC5681] (and [RFC6675]). Note that other mechanisms, such as Proportional Rate Reduction [RFC6937], can be used to reduce the sending rate during loss recovery more gradually. The parameter __cubic_ SHOULD be set to 0.7, which is different from the multiplicative decrease factor used in [RFC5681] (and [RFC6675]) during fast recovery.

In Figure 5, _flight_size_ is the amount of outstanding data in the network, as defined in [RFC5681]. Note that a rate-limited application with idle periods or periods when unable to send at the full rate permitted by _cwnd_ may easily encounter notable variations in the volume of data sent from one RTT to another, resulting in _flight_size_ that is significantly less than _cwnd_ on a congestion event. This may decrease _cwnd_ to a much lower value than necessary. To avoid suboptimal performance with such applications, some implementations of CUBIC use _cwnd_ instead of _flight_size_ to calculate the new _ssthresh_ in Figure 5. Alternatively, the mechanisms described in [RFC7661] may also be adopted to mitigate this issue.

```
            flight_size *       // new ssthresh
ssthresh =                 cubic

            /max(ssthresh, 2)    // reduction on packet loss, cwnd is at least 2
MSS
            |
cwnd =      <
            |max(ssthresh, 1)    // reduction on ECE, cwnd is at least 1 MSS
            \

            max(ssthresh, 2)     // ssthresh is at least 2 MSS
ssthresh =
```

Figure 5

A side effect of setting __cubic_ to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of __cubic_ could result in faster convergence, it will make the analysis of CUBIC much harder.

Note that CUBIC will continue to reduce _cwnd_ in response to congestion events due to ECN-Echo ACKs until it reaches a value of 1 MSS. If congestion persists, a sender with a _cwnd_ of 1 MSS needs

to reduce its sending rate even further. It can achieve that by using a retransmission timer with exponential backoff, as described in [RFC3168].

4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism SHOULD be implemented.

With Fast Convergence, when a congestion event occurs, we update _W_max_ as follows, before the window reduction as described in Section 4.6.

```
       /
       |       1 +
       |            cubic if cwnd < W    and fast convergence is enabled,
       |cwnd * ----------            max
       |            2
W    = <
 max   |                  further reduce W
       |                                 max
       |
       |                  otherwise, remember cwnd before reduction
       \cwnd
```

At a congestion event, if the current _cwnd_ is less than _W_max_, this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. Then we allow this flow to release more bandwidth by reducing _W_max_ further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced _W_max_ forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence SHOULD be disabled.

4.8. Timeout

In case of a timeout, CUBIC follows Reno to reduce _cwnd_ [RFC5681], but sets _ssthresh_ using __cubic_ (same as in Section 4.6) in a way that is different from Reno TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using Figure 1, where _t_ is the elapsed time since the beginning of the current congestion avoidance, _K_ is set to 0, and _W_max_ is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the Reno-friendly region, _W_est_ SHOULD be set to the congestion window size at the beginning of the current congestion avoidance.

4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC's performance significantly.

When there is a congestion event, a CUBIC implementation SHOULD save the current value of the following variables before the congestion window reduction.

```
prior_cwnd = cwnd

prior_ssthresh = ssthresh

prior_W    = W
       max    max

prior_K = K

prior_epoch      = epoch
            start        start

prior_W_{est} = W
                 est
```

CUBIC MAY implement an algorithm to detect spurious retransmissions, such as Forward RTO-Recovery [RFC5682]. Experimental alternatives include DSACK [RFC3708] and Eifel [RFC3522]. Once a spurious congestion event is detected, CUBIC SHOULD restore the original values of above-mentioned variables as follows if the current _cwnd_ is lower than _prior_cwnd_. Restoring the original values ensures that CUBIC's performance is similar to what it would be without spurious losses.

```
                                        \
            cwnd = prior_cwnd           |
                                        |
            ssthresh = prior_ssthresh   |
                                        |
            W    = prior_W              |
             max          max           |
                                        >if cwnd < prior_cwnd
            K = prior_K                 |
                                        |
            epoch      = prior_epoch    |
                 start              start|
                                        |
            W    = prior_W              |
             est          est           /
```

In rare cases, when the detection happens long after a spurious loss event and the current _cwnd_ is already higher than _prior_cwnd_, CUBIC SHOULD continue to use the current and the most recent values of these variables.

4.10. Slow Start

CUBIC MUST employ a slow-start algorithm, when _cwnd_ is no more than _ssthresh_. In general, CUBIC SHOULD use the HyStart++ slow start algorithm [I-D.ietf-tcpm-hystartplusplus], or MAY use the Reno TCP slow start algorithm [RFC5681] in the rare cases when HyStart++ is not suitable. Experimental alternatives include hybrid slow start [HR08], a predecessor to HyStart++ that some CUBIC implementations have used as the default for the last decade, and limited slow start [RFC3742]. Whichever start-up algorithm is used, work might be needed to ensure that the end of slow start and the first multiplicative decrease of congestion avoidance work well together.

When CUBIC uses HyStart++ [I-D.ietf-tcpm-hystartplusplus], it may exit the first slow start without incurring any packet loss and thus _W_max_ is undefined. In this special case, CUBIC switches to congestion avoidance and increases its congestion window size using Figure 1, where _t_ is the elapsed time since the beginning of the current congestion avoidance, _K_ is set to 0, and _W_max_ is set to the congestion window size at the beginning of the current congestion avoidance stage.

5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

With a deterministic loss model where the number of packets between two successive packet losses is always _1/p_, CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

$$AVG_W_{cubic} = \sqrt[4]{\frac{\sqrt[4]{C * (3 + \beta_{cubic})}}{4 * (1 - \beta_{cubic})}} * \frac{\sqrt[4]{RTT^3}}{\sqrt[4]{p^3}}$$

Figure 6

With __cubic_ set to 0.7, the above formula reduces to:

$$AVG_W_{cubic} = \sqrt[4]{\frac{\sqrt{C * 3.7}}{1.2}} * \frac{\sqrt[4]{RTT^3}}{\sqrt[4]{p^3}}$$

Figure 7

We will determine the value of _C_ in the following subsection using Figure 7.

5.1. Fairness to Reno

In environments where Reno is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

Reno performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)

2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to Reno in the above two types of networks. The following two tables show the average window sizes of Reno TCP, HSTCP, and CUBIC TCP. The average window sizes of Reno TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 7 and the CUBIC Reno-friendly region for three different values of _C_.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	59
1.0e-04	120	263	120	187	333
1.0e-05	379	1795	593	1054	1874
1.0e-06	1200	12280	3332	5926	10538
1.0e-07	3795	83981	18740	33325	59261
1.0e-08	12000	574356	105383	187400	333250

Table 1: Reno TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.1 seconds. The average window size is in MSS-sized segments.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	38
1.0e-04	120	263	120	120	120
1.0e-05	379	1795	379	379	379
1.0e-06	1200	12280	1200	1200	1874
1.0e-07	3795	83981	3795	5926	10538
1.0e-08	12000	574356	18740	33325	59261

Table 2: Reno TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.01 seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three _C_ values is more friendly to Reno TCP than HSTCP, especially in networks with a short _RTT_ where Reno TCP performs reasonably well. For example, in a network with _RTT_ = 0.01 seconds and p=10^-6, Reno TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then Reno TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with _C_=0.04 or _C_=0.4 achieves exactly the same rate as Reno TCP, whereas HSTCP is about ten times more aggressive than Reno TCP.

We can see that _C_ determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Reno TCP, if the value of _C_ is lower. However, we do not recommend setting _C_ to a very low value like 0.04, since CUBIC with a low _C_ cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, we find _C_=0.4 gives a good balance between Reno-friendliness and aggressiveness of window increase. Therefore, _C_ SHOULD be set to 0.4. With _C_ set to 0.4, Figure 7 is reduced to:

```
                                   ____
                                4 /   3
                                |/ RTT
AVG_W      = 1.054 * -------
     cubic                        __
                                 4 / 3
                                 |/ p
```

Figure 8

Figure 8 is then used in the next subsection to show the scalability of CUBIC.

5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Reno for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, Reno TCP requires a packet loss rate of 2.0e-10, while CUBIC TCP requires a packet loss rate of 2.9e-8.

Throughput (Mbps)	Average W	Reno P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

Table 3: Required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an _RTT_ of 0.1 seconds.

Our test results in [HKLRX06] indicate that CUBIC uses the spare bandwidth left unused by existing Reno TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of Reno in fast and long-distance networks.

5.4. Investigating a Range of Environments

There is decade-long deployment experience with CUBIC on the Internet. CUBIC has also been extensively studied by using both NS-2 simulation and testbed experiments, covering a wide range of network environments. More information can be found in [HKLRX06].

Same as Reno, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Reno in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Reno and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to mitigate the risk to some extent and reduce the packet queuing delay. Also, in large-BDP networks after a congestion event, CUBIC, due its cubic window increase function, recovers quickly to the highest link utilization

point. This means that link utilization is less sensitive to an active queue management (AQM) target that is lower than the amplitude of the whole sawtooth.

Similar to Reno, the performance of CUBIC as a loss-based congestion control algorithm suffers in networks where a packet loss is not a good indication of bandwidth utilization, such as wireless or mobile networks [LIU16].

5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Reno, since CUBIC modifies only the window adjustment algorithm of Reno. Thus, it does not modify the ACK clocking and timeout behaviors of Reno.

CUBIC also satisfies the "full backoff" requirement as described in [RFC5033]. After reducing the sending rate to one packet per RTT in response to congestion events due to ECN-Echo ACKs, CUBIC then exponentially increases the transmission timer for each packet retransmission while congestion persists.

5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link.

5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC design.

5.8. Behavior for Application-Limited Flows

CUBIC does not increase its congestion window size if a flow is currently limited by the application instead of the congestion window. Section 4.2 requires that _t_ in Figure 1 does not include application-limited periods, such as idle periods, otherwise W_cubic(_t_) might be very high after restarting from these periods.

5.9. Responses to Sudden or Transient Events

If there is a sudden increase in capacity, e.g., due to variable radio capacity, a routing change, or a mobility event, CUBIC is designed to utilize the newly available capacity faster than Reno.

On the other hand, if there is a sudden decrease in capacity, CUBIC reduces more slowly than Reno. This remains true whether or not CUBIC is in Reno-friendly mode and whether or not fast convergence is enabled.

5.10. Incremental Deployment

CUBIC requires only changes to the congestion control at the sender, and it does not require any changes at receivers. That is, a CUBIC sender works correctly with Reno receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

6. Security Considerations

CUBIC makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[I-D.ietf-tcpm-hystartplusplus]
Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-hystartplusplus-03, 25 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-hystartplusplus-03>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/rfc/rfc3168>.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/rfc/rfc5033>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/rfc/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/rfc/rfc5681>.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <https://www.rfc-editor.org/rfc/rfc5682>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/rfc/rfc6298>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/rfc/rfc6582>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/rfc/rfc6675>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/rfc/rfc7567>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/rfc/rfc8985>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

8.2. Informative References

[CEHRX07] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic Ordering for Internet Congestion Control and its Applications", IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications, DOI 10.1109/infcom.2007.111, 2007, <https://doi.org/10.1109/infcom.2007.111>.

[FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000, <https://www.icir.org/tfrc/aimd.pdf>.

[GV02] Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, 11 August 2002, <https://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>.

[HKLRX06] Ha, S., Kim, Y., Le, L., Rhee, I., and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants", International Workshop on Protocols for Fast Long-Distance Networks, February 2006, <https://pfld.net/2006/paper/s2_03.pdf>.

[HR08] Ha, S. and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", International Workshop on Protocols for Fast Long-Distance Networks, March 2008, <http://www.hep.man.ac.uk/g/GDARN-IT/pfldnet2008/paper/Sangate_Ha%20Final.pdf>.

[HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", ACM SIGOPS Operating Systems Review Vol. 42, pp. 64-74, DOI 10.1145/1400097.1400105, July 2008, <https://doi.org/10.1145/1400097.1400105>.

[K03] Kelly, T., "Scalable TCP: improving performance in highspeed wide area networks", ACM SIGCOMM Computer Communication Review Vol. 33, pp. 83-91, DOI 10.1145/956981.956989, April 2003, <https://doi.org/10.1145/956981.956989>.

[LIU16] Liu, K. and J. Lee, "On Improving TCP Performance over Mobile Data Networks", IEEE Transactions on Mobile Computing Vol. 15, pp. 2522-2536, DOI 10.1109/tmc.2015.2500227, October 2016, <https://doi.org/10.1109/tmc.2015.2500227>.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/rfc/rfc3465>.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <https://www.rfc-editor.org/rfc/rfc3522>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/rfc/rfc3649>.

[RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <https://www.rfc-editor.org/rfc/rfc3708>.

[RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <https://www.rfc-editor.org/rfc/rfc3742>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/rfc/rfc4960>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/rfc/rfc6937>.

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/rfc/rfc7661>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/rfc/rfc8312>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/rfc/rfc8511>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[SXEZ19] Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical State Space of Real-World TCP Congestion Control Implementations", USENIX NSDI 2019, February 2019, <https://www.usenix.org/system/files/nsdi19-sun.pdf>.

[XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", IEEE INFOCOM 2004, DOI 10.1109/infcom.2004.1354672, March 2004, <https://doi.org/10.1109/infcom.2004.1354672>.

Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [RFC8312].

These individuals suggested improvements to this document:

* Bob Briscoe

* Christian Huitema

* Gorry Fairhurst

* Jonathan Morton

* Juhamatti Kuusisaari

* Junho Choi

* Markku Kojo

* Martin Thomson

* Matt Olson

* Michael Welzl

* Mirja Kuehlewind

* Mohit P. Tahiliani

* Neal Cardwell

* Praveen Balasubramanian

* Richard Scheffenegger

* Rod Grimes

* Tom Henderson

* Tom Petch

* Wesley Rosenblum

* Yoshifumi Nishida

* Yuchung Cheng

Appendix B. Evolution of CUBIC

B.1. Since draft-ietf-tcpm-rfc8312bis-04

* Fix incorrect math (#106 (https://github.com/NTAP/rfc8312bis/issues/106))

* Update RFC5681 (#99 (https://github.com/NTAP/rfc8312bis/issues/99))

* Rephrase text around algorithmic alternatives, add HyStart++ (#85 (https://github.com/NTAP/rfc8312bis/issues/85), #86 (https://github.com/NTAP/rfc8312bis/issues/86), #90 (https://github.com/NTAP/rfc8312bis/issues/90))

* Clarify what we mean by "new ACK" and use it in the text in more places. (#101 (https://github.com/NTAP/rfc8312bis/issues/101))

* Rewrite the Responses to Sudden or Transient Events section (#98 (https://github.com/NTAP/rfc8312bis/issues/98))

* Remove confusing text about _cwnd_start_ in Section 4.2 (#100 (https://github.com/NTAP/rfc8312bis/issues/100))

* Change terminology from "AIMD" to "Reno" (#108 (https://github.com/NTAP/rfc8312bis/issues/108))

* Moved MUST NOT from app-limited section to main cubic AI section (#97 (https://github.com/NTAP/rfc8312bis/issues/97))

* Clarify cwnd decrease during multiplicative decrease (#102 (https://github.com/NTAP/rfc8312bis/issues/102))

* Clarify text around queuing and slow adaptation of CUBIC in wireless environments (#94 (https://github.com/NTAP/rfc8312bis/issues/94))

* Set lower bound of cwnd to 1 MSS and use retransmit timer thereafter (#83 (https://github.com/NTAP/rfc8312bis/issues/83))

* Use FlightSize instead of cwnd to update ssthresh (#114 (https://github.com/NTAP/rfc8312bis/issues/114))

B.2. Since draft-ietf-tcpm-rfc8312bis-03

* Remove reference from abstract (#82 (https://github.com/NTAP/rfc8312bis/pull/82))

B.3. Since draft-ietf-tcpm-rfc8312bis-02

* Description of packet loss rate _p_ (#65 (https://github.com/NTAP/rfc8312bis/issues/65))

* Clarification of TCP Friendly Equation for ABC and Delayed ACK (#66 (https://github.com/NTAP/rfc8312bis/issues/66))

* add applicability to QUIC and SCTP (#61 (https://github.com/NTAP/rfc8312bis/issues/61))

* clarity on setting alpha__aimd_ to 1 (#68 (https://github.com/NTAP/rfc8312bis/issues/68))

* introduce alpha__cubic_ (#64 (https://github.com/NTAP/rfc8312bis/issues/64))

* clarify _cwnd_ growth in convex region (#69 (https://github.com/NTAP/rfc8312bis/issues/69))

* add guidance for using bytes and mention that segments count is decimal (#67 (https://github.com/NTAP/rfc8312bis/issues/67))

* add loss events detected by RACK and QUIC loss detection (#62 (https://github.com/NTAP/rfc8312bis/issues/62))

B.4. Since draft-ietf-tcpm-rfc8312bis-01

* address Michael Scharf’s editorial suggestions. (#59 (https://github.com/NTAP/rfc8312bis/issues/59))

* add "Note to the RFC Editor" about removing underscores

B.5. Since draft-ietf-tcpm-rfc8312bis-00

* use updated xml2rfc with better text rendering of subscripts

B.6. Since draft-eggert-tcpm-rfc8312bis-03

* fix spelling nits

* rename to draft-ietf

* define _W_max_ more clearly

B.7. Since draft-eggert-tcpm-rfc8312bis-02

* add definition for segments_acked and alpha__aimd_. (#47 (https://github.com/NTAP/rfc8312bis/issues/47))

* fix a mistake in _W_max_ calculation in the fast convergence section. (#51 (https://github.com/NTAP/rfc8312bis/issues/51))

* clarity on setting _ssthresh_ and _cwnd_start_ during multiplicative decrease. (#53 (https://github.com/NTAP/rfc8312bis/issues/53))

B.8. Since draft-eggert-tcpm-rfc8312bis-01

* rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. (#38 (https://github.com/NTAP/rfc8312bis/issues/38))

* change introductory text to reflect the significant broader deployment of CUBIC on the Internet. (#39 (https://github.com/NTAP/rfc8312bis/issues/39))

* rephrase introduction to avoid referring to variables that have not been defined yet.

B.9. Since draft-eggert-tcpm-rfc8312bis-00

* acknowledge former co-authors (#15 (https://github.com/NTAP/rfc8312bis/issues/15))

* prevent _cwnd_ from becoming less than two (#7 (https://github.com/NTAP/rfc8312bis/issues/7))

* add list of variables and constants (#5 (https://github.com/NTAP/rfc8312bis/issues/5), #6 (https://github.com/NTAP/rfc8312bis/issues/6))

* update _K_'s definition and add bounds for CUBIC _target_ _cwnd_ [SXEZ19] (#1 (https://github.com/NTAP/rfc8312bis/issues/1), #14 (https://github.com/NTAP/rfc8312bis/issues/14))

* update _W_est_ to use AIMD approach (#20 (https://github.com/NTAP/rfc8312bis/issues/20))

* set alpha__aimd_ to 1 once _W_est_ reaches _W_max_ (#2 (https://github.com/NTAP/rfc8312bis/issues/2))

* add Vidhi as co-author (#17 (https://github.com/NTAP/rfc8312bis/issues/17))

* note for Fast Recovery during _cwnd_ decrease due to congestion event (#11 (https://github.com/NTAP/rfc8312bis/issues/11))

* add section for spurious congestion events (#23 (https://github.com/NTAP/rfc8312bis/issues/23))

* initialize _W_est_ after timeout and remove variable _W_(last_max)_ (#28 (https://github.com/NTAP/rfc8312bis/issues/28))

B.10. Since RFC8312

* converted to Markdown and xml2rfc v3

* updated references (as part of the conversion)

* updated author information

* various formatting changes

* move to Standards Track

B.11. Since the Original Paper

CUBIC has gone through a few changes since the initial release [HRX08] of its algorithm and implementation. Below we highlight the differences between its original paper and [RFC8312].

* The original paper [HRX08] includes the pseudocode of CUBIC implementation using Linux's pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.

* [HRX08] also includes experimental results showing its performance and fairness.

* The definition of beta__cubic_ constant was changed in [RFC8312]. For example, beta__cubic_ in the original paper was the window decrease constant while [RFC8312] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [RFC8312] was beta__cubic_ * _W_max_ while it was (1-beta__cubic_) * _W_max_ in the original paper.

* Its pseudocode used _W_(last_max)_ while [RFC8312] used _W_max_.

* Its AIMD-friendly window was _W_tcp_ while [RFC8312] used _W_est_.

Authors' Addresses

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
United States of America

Email: xu@unl.edu
URI: https://cse.unl.edu/~xu/

Sangtae Ha
University of Colorado at Boulder
Department of Computer Science
Boulder, CO 80309-0430
United States of America

Email: sangtae.ha@colorado.edu
URI: https://netstech.org/sangtaeha/

Injong Rhee
Bowery Farming
151 W 26TH Street, 12TH Floor
New York, NY 10001
United States of America

Email: injongrhee@gmail.com

Vidhi Goel
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: vidhi_goel@apple.com

Lars Eggert (editor)
NetApp
Stenbergintie 12 B
FI-02700 Kauniainen
Finland

Email: lars@eggert.org
URI: https://eggert.org/

TCPM
Internet-Draft
Obsoletes: 8312 (if approved)
Updates: 5681 (if approved)
Intended status: Standards Track
Expires: 4 August 2023

L. Xu
UNL
S. Ha
Colorado
I. Rhee
Bowery
V. Goel
Apple Inc.
L. Eggert, Ed.
NetApp
31 January 2023

CUBIC for Fast and Long-Distance Networks
draft-ietf-tcpm-rfc8312bis-15

Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of a linear congestion window increase function to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting RFC 8312. This also requires updating RFC 5681, to allow for CUBIC's occasionally more aggressive sending behavior.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tcpm-rfc8312bis/.

Discussion of this document takes place on the TCPM Working Group mailing list (mailto:tcpm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tcpm/. Subscribe at https://www.ietf.org/mailman/listinfo/tcpm/.

Source for this draft and an issue tracker can be found at https://github.com/NTAP/rfc8312bis.

Note to the RFC Editor

xml2rfc currently renders in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: https://github.com/ietf-tools/xml2rfc/issues/596)

Also, please manually change "Figure" to "Equation" for all artwork with anchors beginning with "eq" - xml2rfc doesn't seem to be able to do this.

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 https://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 4 August 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 4
2. Conventions 6
3. Design Principles of CUBIC 6

3.1. Principle 1 for the CUBIC Increase Function 7
3.2. Principle 2 for Reno-Friendliness 7
3.3. Principle 3 for RTT Fairness 8
3.4. Principle 4 for the CUBIC Decrease Factor 8
4. CUBIC Congestion Control 9
4.1. Definitions 9
4.1.1. Constants of Interest 9
4.1.2. Variables of Interest 10
4.2. Window Increase Function 11
4.3. Reno-Friendly Region 12
4.4. Concave Region 14
4.5. Convex Region 14
4.6. Multiplicative Decrease 15
4.7. Fast Convergence 16
4.8. Timeout 17
4.9. Spurious Congestion Events 17
4.9.1. Spurious timeout 18
4.9.2. Spurious loss detected by acknowledgments 18
4.10. Slow Start 19
5. Discussion 20
5.1. Fairness to Reno 21
5.2. Using Spare Capacity 23
5.3. Difficult Environments 24
5.4. Investigating a Range of Environments 24
5.5. Protection against Congestion Collapse 24
5.6. Fairness within the Alternative Congestion Control Algorithm 25
5.7. Performance with Misbehaving Nodes and Outside Attackers 25
5.8. Behavior for Application-Limited Flows 25
5.9. Responses to Sudden or Transient Events 25
5.10. Incremental Deployment 26
6. Security Considerations 26
7. IANA Considerations 26
8. References 26
8.1. Normative References 26
8.2. Informative References 28
Appendix A. Acknowledgments 31
Appendix B. Evolution of CUBIC 32
B.1. Since draft-ietf-tcpm-rfc8312bis-14 32
B.2. Since draft-ietf-tcpm-rfc8312bis-13 32
B.3. Since draft-ietf-tcpm-rfc8312bis-12 32
B.4. Since draft-ietf-tcpm-rfc8312bis-11 32
B.5. Since draft-ietf-tcpm-rfc8312bis-10 32
B.6. Since draft-ietf-tcpm-rfc8312bis-09 32
B.7. Since draft-ietf-tcpm-rfc8312bis-08 32
B.8. Since draft-ietf-tcpm-rfc8312bis-07 33
B.9. Since draft-ietf-tcpm-rfc8312bis-06 33

B.10. Since draft-ietf-tcpm-rfc8312bis-05 33
B.11. Since draft-ietf-tcpm-rfc8312bis-04 33
B.12. Since draft-ietf-tcpm-rfc8312bis-03 34
B.13. Since draft-ietf-tcpm-rfc8312bis-02 34
B.14. Since draft-ietf-tcpm-rfc8312bis-01 35
B.15. Since draft-ietf-tcpm-rfc8312bis-00 35
B.16. Since draft-eggert-tcpm-rfc8312bis-03 35
B.17. Since draft-eggert-tcpm-rfc8312bis-02 35
B.18. Since draft-eggert-tcpm-rfc8312bis-01 35
B.19. Since draft-eggert-tcpm-rfc8312bis-00 36
B.20. Since RFC8312 36
B.21. Since the Original Paper 37
Appendix C. Proof of the Average CUBIC Window Size 37
Authors' Addresses 40

1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over classical Reno congestion control [RFC5681]. It is therefore to be regarded as the currently most widely deployed standard for TCP congestion control. CUBIC can also be used for other transport protocols such as QUIC [RFC9000] and SCTP [RFC9260] as a default congestion controller.

The design of CUBIC was motivated by the well-documented problem classical Reno TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HLRX07] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC9260], TFRC [RFC5348], and QUIC congestion control [RFC9002], which use the same linear increase function for window growth. All Reno-style standards and their variants are collectively referred to as "Reno" in this document.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of classical Reno to remedy this problem. Specifically, CUBIC uses a cubic function instead of the linear window increase function of Reno to improve scalability and stability under fast and long-distance networks.

This document updates the specification of CUBIC to include algorithmic improvements based on the Linux, Windows, and Apple implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312]. This requires an update to Section 3 of [RFC5681], which limits the aggressiveness of Reno TCP implementations. Since CUBIC is occasionally more aggressive than the [RFC5681] algorithms, this document updates the first paragraph of Section 3 of [RFC5681], replacing it with a normative reference to guideline (1) in Section 3 of [RFC5033], which allows for CUBIC's behavior as defined in this document.

Specifically, CUBIC may increase the congestion window more aggressively than Reno during the congestion avoidance phase. According to [RFC5681], during congestion avoidance, the sender must not increment cwnd by more than SMSS bytes once per RTT, whereas CUBIC may increase cwnd much more aggressively. Additionally, CUBIC recommends the HyStart++ algorithm [I-D.ietf-tcpm-hystartplusplus] for slow start, which allows for cwnd increases of more than SMSS bytes for incoming acknowledgments during slow start, while this behavior is not allowed as part of [RFC5681] standard.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Reno in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

[RFC5033] documents the IETF's best current practices for specifying new congestion control algorithms, specifically, ones that differ from the general congestion control principles outlined in [RFC2914]. It describes what type of evaluation is expected by the IETF to understand the suitability of a new congestion control algorithm and the process to enable a specification to be approved for widespread deployment in the global Internet.

There are areas in which CUBIC differs from the congestion control algorithms previously published in standards-track RFCs; those changes are specified in this document. However, it is not obvious that these changes go beyond the general congestion control principles outlined in [RFC2914], so the process in [RFC5033] may not apply.

Also, the wide deployment of CUBIC on the Internet was driven by direct adoption in most of the popular operating systems, and did not follow the practices documented in [RFC5033]. However, due to the resulting Internet-scale deployment experience over a long period of time, the IETF has determined that CUBIC may be published as a standards-track specification. This decision by the IETF does not alter the general guidance in [RFC2914].

The following sections first briefly explain the design principles of CUBIC, provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2: To be Reno-friendly, CUBIC is designed to behave like Reno in networks with short RTTs and small bandwidth where Reno performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Reno increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event detected by duplicate ACKs, Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], TCP RACK [RFC8985] or QUIC loss detection [RFC9002], CUBIC remembers the congestion window size at which it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX09]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

3.2. Principle 2 for Reno-Friendliness

CUBIC promotes per-flow fairness to Reno. Note that Reno performs well over paths with small bandwidth-delay products, and only experiences problems when attempting to increase bandwidth utilization on paths with large bandwidth-delay products.

A congestion control algorithm designed to be friendly to Reno on a per-flow basis must increase its congestion window less aggressively in small-BDP networks than in large-BDP networks.

The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than Reno, CUBIC simply follows the window size of Reno to ensure that CUBIC achieves at least the same throughput as Reno in small-BDP networks. The region where CUBIC behaves like Reno is called the "Reno-friendly region".

3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the Reno-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the Reno-friendly region.

This notion of a linear throughput ratio is similar to that of Reno under an asynchronous loss model, where flows with different RTTs have the same packet loss rate but experience loss events at different times. However, under a synchronous loss model, where flows with different RTTs experience loss events at the same time but have different packet loss rates, the throughput ratio of Reno flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04].

CUBIC always ensures a linear throughput ratio independent of the loss environment. This is an improvement over Reno. While there is no consensus on the optimal throughput ratio for different RTT flows, over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of Reno in synchronous loss environments).

3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas Reno uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that HighSpeed TCP (HSTCP) [RFC3649] and other approaches (e.g., [GV02]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. In environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

4. CUBIC Congestion Control

This section discusses how the congestion window is updated during the different stages of the CUBIC congestion controller.

4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Implementations can use bytes to express window sizes, which would require factoring in the maximum segment size wherever necessary and replacing _segments_acked_ with the number of bytes acknowledged in Figure 4.

4.1.1. Constants of Interest

__cubic_: CUBIC multiplicative decrease factor as described in Section 4.6.

__cubic_: CUBIC additive increase factor used in Reno-friendly region as described in Section 4.3.

_C_: constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high-BDP networks. Please see Section 5 for more explanation on how it is set. The unit for _C_ is

$$\frac{\text{segment}}{\text{second}^3}$$

4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:

RTT: Smoothed round-trip time in seconds, calculated as described in [RFC6298].

cwnd: Current congestion window in segments.

ssthresh: Current slow start threshold in segments.

_cwnd_prior_: Size of _cwnd_ in segments at the time of setting _ssthresh_ most recently, either upon exiting the first slow start, or just before _cwnd_ was reduced in the last congestion event.

_W_max_: Size of _cwnd_ in segments just before _cwnd_ was reduced in the last congestion event when fast convergence is disabled (same as _cwnd_prior_ on a congestion event). However, if fast convergence is enabled, _W_max_ may be further reduced based on the current saturation point.

_K_: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to _W_max_.

_t_current_: Current time of the system in seconds.

_t_epoch_: The time in seconds at which the current congestion avoidance stage started.

_cwnd_epoch_: The _cwnd_ at the beginning of the current congestion avoidance stage, i.e., at time _t_epoch_.

W_cubic(_t_): The congestion window in segments at time _t_ in seconds based on the cubic increase function, as described in Section 4.2.

target: Target value of congestion window in segments after the next RTT, that is, W_cubic(_t_ + _RTT_), as described in Section 4.2.

_W_est_: An estimate for the congestion window in segments in the Reno-friendly region, that is, an estimate for the congestion window of Reno.

_segments_acked_: Number of MSS-sized segments acked when a "new ACK" is received, i.e., an ACK that cumulatively acknowledges the delivery of new data. This number will be a decimal value when a new ACK acknowledges an amount of data that is not MSS-sized. Specifically, it can be less than 1 when a new ACK acknowledges a segment smaller than the MSS.

4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Reno by increasing the congestion window only at the reception of a new ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].

During congestion avoidance, after a congestion event is detected by mechanisms described in Section 3.1, CUBIC uses a window increase function different from Reno.

CUBIC uses the following window increase function:

$$W_{cubic}(t) = C * (t - K)^3 + W_{max}$$

Figure 1

where _t_ is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

$$t = t_{current} - t_{epoch}$$

and where _t_epoch_ is the time at which the current congestion avoidance stage starts. _K_ is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to _W_max_ if there are no further congestion events and is calculated using the following equation:

$$K = \sqrt[3]{\frac{W_{max} - cwnd_{epoch}}{C}}$$

Figure 2

where _cwnd_epoch_ is the congestion window at the beginning of the current congestion avoidance stage.

Upon receiving a new ACK during congestion avoidance, CUBIC computes the _target_ congestion window size after the next _RTT_ using Figure 1 as follows, where _RTT_ is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC's congestion window increase rate is non-decreasing and is less than the increase rate of slow start [SXEZ19].

```
              cwnd            if  W     (t + RTT) < cwnd
                                   cubic
              1.5 * cwnd      if  W     (t + RTT) > 1.5 * cwnd
     target =                      cubic
              W     (t + RTT) otherwise
               cubic
```

The elapsed time _t_ in Figure 1 MUST NOT include periods during which _cwnd_ has not been updated due to application-limited behavior (see Section 5.8).

Depending on the value of the current congestion window size _cwnd_, CUBIC runs in three different regions:

1. The Reno-friendly region, which ensures that CUBIC achieves at least the same throughput as Reno.

2. The concave region, if CUBIC is not in the Reno-friendly region and _cwnd_ is less than _W_max_.

3. The convex region, if CUBIC is not in the Reno-friendly region and _cwnd_ is greater than _W_max_.

To summarize, CUBIC computes both W_cubic(_t_) and _W_est_ (see Section 4.3) on receiving a new ACK in congestion avoidance and chooses the larger of the two values.

The next sections describe the exact actions taken by CUBIC in each region.

4.3. Reno-Friendly Region

Reno performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the Reno-friendly region to achieve at least the same throughput as Reno.

The Reno-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of (segments per _RTT_) and a multiplicative factor of , denoted by AIMD(,). _p_ is the packet loss rate. Specifically, the average congestion window size of AIMD(,) can be calculated using Figure 3.

```
                                  * (1 + )
AVG_AIMD(, ) =
                         2 * (1 - ) * p
```

Figure 3

By the same analysis, to achieve a similar average window size as Reno that uses AIMD(1, 0.5), must be equal to,

```
                              1 -
                          3 *
                              1 +
```

Thus, CUBIC uses Figure 4 to estimate the window size _W_est_ in the Reno-friendly region with

```
                                    1 -
                                         cubic
                            = 3 *
                      cubic       1 +
                                         cubic
```

which achieves approximately the same average window size as Reno in many cases. The model used to calculate __cubic_ is not absolutely precise, but analysis and simulation in [AIMD-friendliness], as well as over a decade of experience with CUBIC in the public Internet, show that this approach produces acceptable levels of rate fairness between CUBIC and Reno flows. Also, no significant drawbacks of the model have been reported. However, it would be beneficial to see continued detailed analysis on it. When receiving a new ACK in congestion avoidance (where _cwnd_ could be greater than or less than _W_max_), CUBIC checks whether W_cubic(_t_) is less than _W_est_. If so, CUBIC is in the Reno-friendly region and _cwnd_ SHOULD be set to _W_est_ at each reception of a new ACK.

_W_est_ is set equal to _cwnd_epoch_ at the start of the congestion avoidance stage. After that, on every new ACK, _W_est_ is updated using Figure 4. Note that this equation uses _segments_acked_ and _cwnd_ is measured in segments. An implementation that measures _cwnd_ in bytes should adjust the equation accordingly using number

of acknowledged bytes and MSS. Also note that this equation works for connections with enabled or disabled Delayed ACKs [RFC5681], as _segments_acked_ will be different based on the segments actually acknowledged by a new ACK.

$$W_{est} = W_{est} + \frac{segments_acked}{cubic} * \frac{}{cwnd}$$

Figure 4

Once _W_est_ has grown to reach the _cwnd_ at the time of most recently setting _ssthresh_, that is, _W_est_ >= _cwnd_prior_, the sender SHOULD set __cubic_ to 1 to ensure that it can achieve the same congestion window increment rate as Reno, which uses AIMD(1, 0.5).

The next two sections assume that CUBIC is not in the Reno-friendly region and uses the window increase function described in Section 4.2. Although _cwnd_ is incremented in the same way for both concave and convex regions, they are discussed separately to analyze and understand the difference between the two regions.

4.4. Concave Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and _cwnd_ is less than _W_max_, then CUBIC is in the concave region. In this region, _cwnd_ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where _target_ is calculated as described in Section 4.2.

4.5. Convex Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and _cwnd_ is larger than or equal to _W_max_, then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

Unless it is overridden by the AIMD window increase, CUBIC is very careful in this region. The convex profile aims to increase the window very slowly at the beginning when _cwnd_ is around _W_max_ and then gradually increases its rate of increase. This region is also called the "maximum probing phase", since CUBIC is searching for a new _W_max_. In this region, _cwnd_ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where _target_ is calculated as described in Section 4.2.

4.6. Multiplicative Decrease

When a congestion event is detected by mechanisms described in Section 3.1, CUBIC updates _W_max_ and reduces _cwnd_ and _ssthresh_ immediately as described below. In case of packet loss, the sender MUST reduce _cwnd_ and _ssthresh_ immediately upon entering loss recovery, similar to [RFC5681] (and [RFC6675]). Note that other mechanisms, such as Proportional Rate Reduction [RFC6937], can be used to reduce the sending rate during loss recovery more gradually. The parameter __cubic_ SHOULD be set to 0.7, which is different from the multiplicative decrease factor used in [RFC5681] (and [RFC6675]) during fast recovery.

In Figure 5, _flight_size_ is the amount of outstanding (unacknowledged) data in the network, as defined in [RFC5681]. Note that a rate-limited application with idle periods or periods when unable to send at the full rate permitted by _cwnd_ could easily encounter notable variations in the volume of data sent from one RTT to another, resulting in _flight_size_ that is significantly less than _cwnd_ when there is a congestion event. The congestion response would therefore decrease _cwnd_ to a much lower value than necessary. To avoid such suboptimal performance, the mechanisms described in [RFC7661] can be used. These describe how to manage and use _cwnd_ and _ssthresh_ during a rate-limited Interval, and how to update _cwnd_ and _ssthresh_ after congestion has been detected. The mechanism defined in [RFC7661] is safe to use even when _cwnd_ is greater than the receive window, because it validates _cwnd_ based on the amount of data acknowledged by the network in an RTT, which implicitly accounts for the allowed receive window.

Some implementations of CUBIC currently use _cwnd_ instead of _flight_size_ when calculating a new _ssthresh_. Implementations that use _cwnd_ MUST use other measures to prevent _cwnd_ from growing when the volume of bytes in flight is smaller than _cwnd_. This also

effectively avoids _cwnd_ from growing beyond the receive window. Such measures are important to prevent a CUBIC sender from using an arbitrarily high cwnd _value_ when calculating new values for _ssthresh_ and _cwnd_ when congestion is detected. This might not be as robust as the mechanisms described in [RFC7661].

A QUIC sender that uses _cwnd_ to calculate new values for _cwnd_ and _ssthresh_ after detecting a congestion event is REQUIRED to apply similar mechanisms [RFC9002].

```
ssthresh =  flight_size *       new  ssthresh
                            cubic
cwnd      = cwnd                 save  cwnd
    prior

            max(ssthresh, 2)    reduction on loss, cwnd is at least 2 MSS
cwnd =      max(ssthresh, 1)    reduction on ECE, cwnd is at least 1 MSS
ssthresh =  max(ssthresh, 2)     ssthresh is at least 2 MSS
```

Figure 5

A side effect of setting __cubic_ to a value bigger than 0.5 is that packet loss can happen for more than one round-trip in certain cases, but it can work efficiently in other cases, for example, when HyStart++ is used along with CUBIC or when the sending rate is limited by the application. While a more adaptive setting of __cubic_ could help limit packet loss to a single round, it would require detailed analyses and large-scale evaluations to validate such algorithms.

Note that CUBIC MUST continue to reduce _cwnd_ in response to congestion events due to ECN-Echo ACKs until it reaches a value of 1 MSS. If congestion events indicated by ECN-Echo ACKs persist, a sender with a _cwnd_ of 1 MSS MUST reduce its sending rate even further. It can achieve that by using a retransmission timer with exponential backoff, as described in [RFC3168].

4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism SHOULD be implemented.

With Fast Convergence, when a congestion event occurs, _W_max_ is updated as follows, before the window reduction described in Section 4.6.

```
                1 +
                     cubic
       cwnd *  if  cwnd < W     and fast convergence enabled,
W    =            2                  max
 max                      further reduce  W
                                           max
       cwnd             otherwise, remember cwnd before reduction
```

At a congestion event, if the current _cwnd_ is less than _W_max_, this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. This flow can then release more bandwidth by reducing _W_max_ further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced _W_max_ forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence SHOULD be disabled.

4.8. Timeout

In case of a timeout, CUBIC follows Reno to reduce _cwnd_ [RFC5681], but sets _ssthresh_ using __cubic_ (same as in Section 4.6) in a way that is different from Reno TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using Figure 1, where _t_ is the elapsed time since the beginning of the current congestion avoidance, _K_ is set to 0, and _W_max_ is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the Reno-friendly region, _W_est_ SHOULD be set to the congestion window size at the beginning of the current congestion avoidance.

4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window

reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC's performance significantly.

For TCP, there are two types of spurious events - spurious timeouts and spurious fast retransmits. In case of QUIC, there are no spurious timeouts as the loss is only detected after receiving an ACK.

4.9.1. Spurious timeout

An implementation MAY detect spurious timeouts based on the mechanisms described in Forward RTO-Recovery [RFC5682]. Experimental alternatives include Eifel [RFC3522]. When a spurious timeout is detected, a TCP implementation MAY follow the response algorithm described in [RFC4015] to restore the congestion control state and adapt the retransmission timer to avoid further spurious timeouts.

4.9.2. Spurious loss detected by acknowledgments

Upon receiving an ACK, a TCP implementation MAY detect spurious losses either using TCP Timestamps or via D-SACK[RFC2883]. Experimental alternatives include Eifel detection algorithm [RFC3522] which uses TCP Timestamps and DSACK based detection [RFC3708] which uses DSACK information. A QUIC implementation can easily determine a spurious loss if a QUIC packet is acknowledged after it has been marked as lost and the original data has been retransmitted with a new QUIC packet.

This section specifies a simple response algorithm when a spurious loss is detected by acknowledgments. Implementations would need to carefully evaluate the impact of using this algorithm in different environments that may experience sudden change in available capacity (e.g., due to variable radio capacity, a routing change, or a mobility event).

When a packet loss is detected via acknowledgments, a CUBIC implementation MAY save the current value of the following variables before the congestion window is reduced.

```
undo_cwnd =      cwnd
undo_cwnd      = cwnd
         prior       prior
undo_ssthresh =  ssthresh
undo_W    =      W
      max         max
undo_K =         K
undo_t      =    t
      epoch       epoch
undo_W    =      W
      est         est
```

Once the previously declared packet loss is confirmed to be spurious, CUBIC MAY restore the original values of the above-mentioned variables as follows if the current _cwnd_ is lower than _cwnd_prior_. Restoring the original values ensures that CUBIC's performance is similar to what it would be without spurious losses.

```
cwnd =      undo_cwnd
cwnd      = undo_cwnd
    prior            prior
ssthresh =  undo_ssthresh
W    =      undo_W
 max             max      if cwnd < cwnd
K =         undo_K                       prior
t      =    undo_t
 epoch           epoch
W    =      undo_W
 est             est
```

In rare cases, when the detection happens long after a spurious loss event and the current _cwnd_ is already higher than _cwnd_prior_, CUBIC SHOULD continue to use the current and the most recent values of these variables.

4.10. Slow Start

CUBIC MUST employ a slow-start algorithm, when _cwnd_ is no more than _ssthresh_. In general, CUBIC SHOULD use the HyStart++ slow start algorithm [I-D.ietf-tcpm-hystartplusplus], or MAY use the Reno TCP slow start algorithm [RFC5681] in the rare cases when HyStart++ is not suitable. Experimental alternatives include hybrid slow start [HR11], a predecessor to HyStart++ that some CUBIC implementations have used as the default for the last decade, and limited slow start [RFC3742]. Whichever start-up algorithm is used, work might be needed to ensure that the end of slow start and the first multiplicative decrease of congestion avoidance work well together.

When CUBIC uses HyStart++ [I-D.ietf-tcpm-hystartplusplus], it may exit the first slow start without incurring any packet loss and thus _W_max_ is undefined. In this special case, CUBIC sets _cwnd_prior = cwnd_ and switches to congestion avoidance. It then increases its congestion window size using Figure 1, where _t_ is the elapsed time since the beginning of the current congestion avoidance, _K_ is set to 0, and _W_max_ is set to the congestion window size at the beginning of the current congestion avoidance stage.

5. Discussion

This section further discusses the safety features of CUBIC following the guidelines specified in [RFC5033].

With a deterministic loss model where the number of packets between two successive packet losses is always _1/p_, CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC (see Appendix C) can be obtained by the following function:

```
                                4
                            C * (3 +      )        3
                        4            cubic      RTT
            AVG_W      =     *
                 cubic     4 * (1 -      )     4
                                     cubic        3
                                                   p
```

Figure 6

With __cubic_ set to 0.7, the above formula reduces to:

```
                                            4
                                        3
                              4 C * 3.7     RTT
                 AVG_W      =    *
                      cubic      1.2       4
                                               3
                                             p
```

Figure 7

The following subsection will determine the value of _C_ using Figure 7.

5.1. Fairness to Reno

In environments where Reno is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

Reno performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)

2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to Reno in the above two types of networks. The following two tables show the average window sizes of Reno TCP, HSTCP, and CUBIC TCP. The average window sizes of Reno TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 7 and the CUBIC Reno-friendly region for three different values of _C_.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	59
1.0e-04	120	263	120	187	333
1.0e-05	379	1795	593	1054	1874
1.0e-06	1200	12280	3332	5926	10538
1.0e-07	3795	83981	18740	33325	59261
1.0e-08	12000	574356	105383	187400	333250

Table 1: Reno TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.1 seconds. The average window size is in MSS-sized segments.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	38
1.0e-04	120	263	120	120	120
1.0e-05	379	1795	379	379	379
1.0e-06	1200	12280	1200	1200	1874
1.0e-07	3795	83981	3795	5926	10538
1.0e-08	12000	574356	18740	33325	59261

Table 2: Reno TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.01 seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three _C_ values is more friendly to Reno TCP than HSTCP, especially in networks with a short _RTT_ where Reno TCP performs reasonably well. For example, in a network with _RTT_ = 0.01 seconds and p=10^-6, Reno TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then Reno TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with _C_=0.04 or _C_=0.4 achieves exactly the same rate as Reno TCP, whereas HSTCP is about ten times more aggressive than Reno TCP.

_C_ determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Reno TCP, if the value of _C_ is lower. However, it is NOT RECOMMENDED to set _C_ to a very low value like 0.04, since CUBIC with a low _C_ cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, _C_=0.4 seems to give a good balance between Reno-friendliness and aggressiveness of window increase. Therefore, _C_ SHOULD be set to 0.4. With _C_ set to 0.4, Figure 7 is reduced to:

$$AVG_W_{cubic} = 1.054 * \frac{RTT^{4/3}}{p^{3/4}}$$

Figure 8

Figure 8 is then used in the next subsection to show the scalability of CUBIC.

5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Reno for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, Reno TCP requires a packet loss rate of 2.0e-10, while CUBIC TCP requires a packet loss rate of 2.9e-8.

Throughput (Mbps)	Average W	Reno P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

Table 3: Required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput, with 1500-byte packets and an _RTT_ of 0.1 seconds.

The test results in [HLRX07] indicate that, in typical cases with a degree of background traffic, CUBIC uses the spare bandwidth left unused by existing Reno TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of Reno in fast and long-distance networks.

5.4. Investigating a Range of Environments

CUBIC has been extensively studied using simulations, testbed emulations, Internet experiments, and Internet measurements, covering a wide range of network environments [HLRX07][H16][CEHRX09][HR11][BSCLU13][LBEWK16]. They have convincingly demonstrated that CUBIC delivers substantial benefits over classical Reno congestion control [RFC5681].

Same as Reno, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Reno in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Reno and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to mitigate the risk to some extent and reduce the packet queuing delay. Also, in large-BDP networks after a congestion event, CUBIC, due its cubic window increase function, recovers quickly to the highest link utilization point. This means that link utilization is less sensitive to an active queue management (AQM) target that is lower than the amplitude of the whole sawtooth.

Similar to Reno, the performance of CUBIC as a loss-based congestion control algorithm suffers in networks where a packet loss is not a good indication of bandwidth utilization, such as wireless or mobile networks [LIU16].

5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Reno, since CUBIC modifies only the window adjustment algorithm of Reno. Thus, it does not modify the ACK clocking and timeout behaviors of Reno.

CUBIC also satisfies the "full backoff" requirement as described in [RFC5033]. After reducing the sending rate to one packet per RTT in response to congestion events due to ECN-Echo ACKs, CUBIC then exponentially increases the transmission timer for each packet retransmission while congestion persists.

5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link. The convergence time depends on the network environments (e.g., bandwidth, RTT) and the level of statistical multiplexing, as mentioned in Section 3.4.

5.7. Performance with Misbehaving Nodes and Outside Attackers

CUBIC does not introduce new entities or signals, so its vulnerability to misbehaving nodes or attackers is unchanged from Reno.

5.8. Behavior for Application-Limited Flows

A flow is application-limited if it is currently sending less than what is allowed by the congestion window. This can happen if the flow is limited by either the sender application or the receiver application (via the receiver advertised window) and thus sends less data than what is allowed by the sender's congestion window.

CUBIC does not increase its congestion window if a flow is application-limited. Section 4.2 requires that _t_ in Figure 1 does not include application-limited periods, such as idle periods, otherwise W_cubic(_t_) might be very high after restarting from these periods.

5.9. Responses to Sudden or Transient Events

If there is a sudden increase in capacity, e.g., due to variable radio capacity, a routing change, or a mobility event, CUBIC is designed to utilize the newly available capacity faster than Reno.

On the other hand, if there is a sudden decrease in capacity, CUBIC reduces more slowly than Reno. This remains true regardless of whether CUBIC is in Reno-friendly mode and regardless of whether fast convergence is enabled.

5.10. Incremental Deployment

CUBIC requires only changes to the congestion control at the sender, and it does not require any changes at receivers. That is, a CUBIC sender works correctly with Reno receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

6. Security Considerations

CUBIC makes no changes to the underlying security of a transport protocol and inherits the general security concerns described in [RFC5681]. Specifically, changing the window computation on the sender may allow an attacker, through dropping or injecting ACKs (as described in [RFC5681]), to either force the CUBIC implementation to reduce its bandwidth, or to convince it that there is no congestion when congestion does exist, and use the CUBIC implementation as an attack vector against other hosts. These attacks are not new to CUBIC and are inherently part of any transport protocol like TCP.

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[I-D.ietf-tcpm-hystartplusplus]
Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-hystartplusplus-13, 30 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-hystartplusplus-13>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <https://www.rfc-editor.org/rfc/rfc2883>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/rfc/rfc2914>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/rfc/rfc3168>.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, DOI 10.17487/RFC4015, February 2005, <https://www.rfc-editor.org/rfc/rfc4015>.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/rfc/rfc5033>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/rfc/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/rfc/rfc5681>.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <https://www.rfc-editor.org/rfc/rfc5682>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/rfc/rfc6298>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/rfc/rfc6582>.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/rfc/rfc6675>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/rfc/rfc7567>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/rfc/rfc8985>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

8.2. Informative References

[AIMD-friendliness] Briscoe, B. and O. Albisser, "Friendliness between AIMD Algorithms", RFC Editor, please replace this URL with the permanent arXiv one , 8 August 2022, <https://raw.githubusercontent.com/bbriscoe/cubic-reno/main/creno_tr.pdf>.

[BSCLU13] Belhareth, S., Sassatelli, L., Collange, D., Lopez-Pacheco, D., and G. Urvoy-Keller, "Understanding TCP cubic performance in the cloud: A mean-field approach", 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), DOI 10.1109/cloudnet.2013.6710576, November 2013, <https://doi.org/10.1109/cloudnet.2013.6710576>.

[CEHRX09] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic convex ordering for multiplicative decrease internet congestion control", Computer Networks vol. 53, no. 3, pp. 365-381, DOI 10.1016/j.comnet.2008.10.012, February 2009, <https://doi.org/10.1016/j.comnet.2008.10.012>.

[FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000, <https://www.icir.org/tfrc/aimd.pdf>.

[GV02] Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, 11 August 2002, <https://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>.

[H16] Ha, S., "Simulation, Testbed, and Deployment Testing Results of CUBIC", 3 November 2016, <https://web.archive.org/web/20161118125842/http://netsrv.csc.ncsu.edu/wiki/index.php/TCP_Testing>.

[HLRX07] Ha, S., Le, L., Rhee, I., and L. Xu, "Impact of background traffic on performance of high-speed TCP variant protocols", Computer Networks vol. 51, no. 7, pp. 1748-1762, DOI 10.1016/j.comnet.2006.11.005, May 2007, <https://doi.org/10.1016/j.comnet.2006.11.005>.

[HR11] Ha, S. and I. Rhee, "Taming the elephants: New TCP slow start", Computer Networks vol. 55, no. 9, pp. 2092-2110, DOI 10.1016/j.comnet.2011.01.014, June 2011, <https://doi.org/10.1016/j.comnet.2011.01.014>.

[HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", ACM SIGOPS Operating Systems Review vol. 42, no. 5, pp. 64-74, DOI 10.1145/1400097.1400105, July 2008, <https://doi.org/10.1145/1400097.1400105>.

[K03] Kelly, T., "Scalable TCP: improving performance in highspeed wide area networks", ACM SIGCOMM Computer Communication Review vol. 33, no. 2, pp. 83-91, DOI 10.1145/956981.956989, April 2003, <https://doi.org/10.1145/956981.956989>.

[LBEWK16] Lukaseder, T., Bradatsch, L., Erb, B., Van Der Heijden, R., and F. Kargl, "A Comparison of TCP Congestion Control Algorithms in 10G Networks", 2016 IEEE 41st Conference on Local Computer Networks (LCN), DOI 10.1109/lcn.2016.121, November 2016, <https://doi.org/10.1109/lcn.2016.121>.

[LIU16] Liu, K. and J. Lee, "On Improving TCP Performance over Mobile Data Networks", IEEE Transactions on Mobile Computing vol. 15, no. 10, pp. 2522-2536, DOI 10.1109/tmc.2015.2500227, October 2016, <https://doi.org/10.1109/tmc.2015.2500227>.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <https://www.rfc-editor.org/rfc/rfc3522>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/rfc/rfc3649>.

[RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <https://www.rfc-editor.org/rfc/rfc3708>.

[RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <https://www.rfc-editor.org/rfc/rfc3742>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/rfc/rfc6937>.

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/rfc/rfc7661>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/rfc/rfc8312>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/rfc/rfc8511>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/rfc/rfc9260>.

[SXEZ19] Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical Congestion Control State Space of Real-World TCP Implementations", IEEE/ACM Transactions on Networking vol. 29, no. 5, pp. 1990-2004, DOI 10.1109/tnet.2021.3078161, October 2021, <https://doi.org/10.1109/tnet.2021.3078161>.

[XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks", IEEE INFOCOM 2004, DOI 10.1109/infcom.2004.1354672, February 2005, <https://doi.org/10.1109/infcom.2004.1354672>.

Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [RFC8312].

These individuals suggested improvements to this document:

* Bob Briscoe
* Christian Huitema
* Gorry Fairhurst
* Jonathan Morton
* Juhamatti Kuusisaari
* Junho Choi
* Markku Kojo
* Martin Duke
* Martin Thomson
* Matt Mathis
* Matt Olson
* Michael Welzl
* Mirja Kühlewind
* Mohit P. Tahiliani
* Neal Cardwell
* Praveen Balasubramanian
* Randall Stewart
* Richard Scheffenegger
* Rod Grimes
* Spencer Dawkins
* Tom Henderson
* Tom Petch
* Wesley Rosenblum
* Yoav Nir
* Yoshifumi Nishida
* Yuchung Cheng

Appendix B. Evolution of CUBIC

B.1. Since draft-ietf-tcpm-rfc8312bis-14

* Specify how security considerations of TCP applies to CUBIC.

* Elaborate differences between RFC 5681 and Cubic.

* Tweak math for better plaintext rendering. (#164 (https://github.com/NTAP/rfc8312bis/pull/164))

B.2. Since draft-ietf-tcpm-rfc8312bis-13

* Add contents of https://cse.unl.edu/~xu/avg_cubic_cwnd.pdf to Appendix C.

* Multiple comments from Martin, define synchronized/asynchronized loss model, clean up 3465 reference, clarficiation for when Cubic is not in Reno-friendly region, referring to proof of avg Cubic window, better text for misbehaving nodes and fix typo in _cwnd_epoch_. (#158 (https://github.com/NTAP/rfc8312bis/pull/158))

B.3. Since draft-ietf-tcpm-rfc8312bis-12

* Fix plaintext version of Figure 5.

B.4. Since draft-ietf-tcpm-rfc8312bis-11

* Fix various nits. (#157 (https://github.com/NTAP/rfc8312bis/pull/157))

B.5. Since draft-ietf-tcpm-rfc8312bis-10

* Improve text related to [RFC7661]. (#149 (https://github.com/NTAP/rfc8312bis/issues/149))

* Made variable naming a bit more consistent. (#156 (https://github.com/NTAP/rfc8312bis/pull/156))

B.6. Since draft-ietf-tcpm-rfc8312bis-09

* Improve text for Reno friendliness, multiplicative decrease and reference to HLRX07. (#152 (https://github.com/NTAP/rfc8312bis/pull/152))

B.7. Since draft-ietf-tcpm-rfc8312bis-08

* Fix the text specifying when alpha_cubic SHOULD be set to 1 to indicate this should happen when cwnd >= cwnd_prior rather than cwnd >= W_max, since these are different in the fast convergence case (#146 (https://github.com/NTAP/rfc8312bis/pull/146))

* Restrict use of _cwnd_ directly on a congestion event (#148 (https://github.com/NTAP/rfc8312bis/pull/148))

B.8. Since draft-ietf-tcpm-rfc8312bis-07

* Document the WG discussion and decision around [RFC5033] and [RFC2914] (#145 (https://github.com/NTAP/rfc8312bis/pull/145))

B.9. Since draft-ietf-tcpm-rfc8312bis-06

* RFC7661 is safe even when cwnd grows beyond rwnd (#143 (https://github.com/NTAP/rfc8312bis/issues/143))

B.10. Since draft-ietf-tcpm-rfc8312bis-05

* Clarify meaning of "application-limited" in Section 5.8 (#137 (https://github.com/NTAP/rfc8312bis/issues/137))

* Create new subsections for spurious timeouts and spurious loss via ACK (#90 (https://github.com/NTAP/rfc8312bis/issues/90))

* Brief discussion of convergence in Section 5.6 (#96 (https://github.com/NTAP/rfc8312bis/issues/96))

* Add more test results to Section 5 and update some references (#91 (https://github.com/NTAP/rfc8312bis/issues/91))

* Change wording around setting ssthresh (#131 (https://github.com/NTAP/rfc8312bis/issues/131))

B.11. Since draft-ietf-tcpm-rfc8312bis-04

* Fix incorrect math (#106 (https://github.com/NTAP/rfc8312bis/issues/106))

* Update RFC5681 (#99 (https://github.com/NTAP/rfc8312bis/issues/99))

* Rephrase text around algorithmic alternatives, add HyStart++ (#85 (https://github.com/NTAP/rfc8312bis/issues/85), #86 (https://github.com/NTAP/rfc8312bis/issues/86), #90 (https://github.com/NTAP/rfc8312bis/issues/90))

* Clarify what we mean by "new ACK" and use it in the text in more places. (#101 (https://github.com/NTAP/rfc8312bis/issues/101))

* Rewrite the Responses to Sudden or Transient Events section (#98 (https://github.com/NTAP/rfc8312bis/issues/98))

* Remove confusing text about _cwnd_epoch_ in Section 4.2 (#100 (https://github.com/NTAP/rfc8312bis/issues/100))

* Change terminology from "AIMD" to "Reno" (#108 (https://github.com/NTAP/rfc8312bis/issues/108))

* Moved MUST NOT from app-limited section to main cubic AI section (#97 (https://github.com/NTAP/rfc8312bis/issues/97))

* Clarify cwnd decrease during multiplicative decrease (#102 (https://github.com/NTAP/rfc8312bis/issues/102))

* Clarify text around queuing and slow adaptation of CUBIC in wireless environments (#94 (https://github.com/NTAP/rfc8312bis/issues/94))

* Set lower bound of cwnd to 1 MSS and use retransmit timer thereafter (#83 (https://github.com/NTAP/rfc8312bis/issues/83))

* Use FlightSize instead of cwnd to update ssthresh (#114 (https://github.com/NTAP/rfc8312bis/issues/114))

B.12. Since draft-ietf-tcpm-rfc8312bis-03

* Remove reference from abstract (#82 (https://github.com/NTAP/rfc8312bis/pull/82))

B.13. Since draft-ietf-tcpm-rfc8312bis-02

* Description of packet loss rate _p_ (#65 (https://github.com/NTAP/rfc8312bis/issues/65))

* Clarification of TCP Friendly Equation for ABC and Delayed ACK (#66 (https://github.com/NTAP/rfc8312bis/issues/66))

* add applicability to QUIC and SCTP (#61 (https://github.com/NTAP/rfc8312bis/issues/61))

* clarity on setting alpha__aimd_ to 1 (#68 (https://github.com/NTAP/rfc8312bis/issues/68))

* introduce alpha__cubic_ (#64 (https://github.com/NTAP/rfc8312bis/issues/64))

* clarify _cwnd_ growth in convex region (#69 (https://github.com/NTAP/rfc8312bis/issues/69))

* add guidance for using bytes and mention that segments count is decimal (#67 (https://github.com/NTAP/rfc8312bis/issues/67))

* add loss events detected by RACK and QUIC loss detection (#62 (https://github.com/NTAP/rfc8312bis/issues/62))

B.14. Since draft-ietf-tcpm-rfc8312bis-01

* address Michael Scharf's editorial suggestions. (#59 (https://github.com/NTAP/rfc8312bis/issues/59))

* add "Note to the RFC Editor" about removing underscores

B.15. Since draft-ietf-tcpm-rfc8312bis-00

* use updated xml2rfc with better text rendering of subscripts

B.16. Since draft-eggert-tcpm-rfc8312bis-03

* fix spelling nits

* rename to draft-ietf

* define _W_max_ more clearly

B.17. Since draft-eggert-tcpm-rfc8312bis-02

* add definition for segments_acked and alpha__aimd_. (#47 (https://github.com/NTAP/rfc8312bis/issues/47))

* fix a mistake in _W_max_ calculation in the fast convergence section. (#51 (https://github.com/NTAP/rfc8312bis/issues/51))

* clarity on setting _ssthresh_ and _cwnd_epoch_ during multiplicative decrease. (#53 (https://github.com/NTAP/rfc8312bis/issues/53))

B.18. Since draft-eggert-tcpm-rfc8312bis-01

* rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. (#38 (https://github.com/NTAP/rfc8312bis/issues/38))

* change introductory text to reflect the significant broader deployment of CUBIC on the Internet. (#39 (https://github.com/NTAP/rfc8312bis/issues/39))

* rephrase introduction to avoid referring to variables that have not been defined yet.

B.19. Since draft-eggert-tcpm-rfc8312bis-00

* acknowledge former co-authors (#15 (https://github.com/NTAP/rfc8312bis/issues/15))

* prevent _cwnd_ from becoming less than two (#7 (https://github.com/NTAP/rfc8312bis/issues/7))

* add list of variables and constants (#5 (https://github.com/NTAP/rfc8312bis/issues/5), #6 (https://github.com/NTAP/rfc8312bis/issues/6))

* update _K_'s definition and add bounds for CUBIC _target_ _cwnd_ [SXEZ19] (#1 (https://github.com/NTAP/rfc8312bis/issues/1), #14 (https://github.com/NTAP/rfc8312bis/issues/14))

* update _W_est_ to use AIMD approach (#20 (https://github.com/NTAP/rfc8312bis/issues/20))

* set alpha__aimd_ to 1 once _W_est_ reaches _W_max_ (#2 (https://github.com/NTAP/rfc8312bis/issues/2))

* add Vidhi as co-author (#17 (https://github.com/NTAP/rfc8312bis/issues/17))

* note for Fast Recovery during _cwnd_ decrease due to congestion event (#11 (https://github.com/NTAP/rfc8312bis/issues/11))

* add section for spurious congestion events (#23 (https://github.com/NTAP/rfc8312bis/issues/23))

* initialize _W_est_ after timeout and remove variable _W_(last_max)_ (#28 (https://github.com/NTAP/rfc8312bis/issues/28))

B.20. Since RFC8312

* converted to Markdown and xml2rfc v3

* updated references (as part of the conversion)

* updated author information

* various formatting changes

* move to Standards Track

B.21. Since the Original Paper

CUBIC has gone through a few changes since the initial release [HRX08] of its algorithm and implementation. This section highlights the differences between the original paper and [RFC8312].

* The original paper [HRX08] includes the pseudocode of CUBIC implementation using Linux's pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.

* [HRX08] also includes experimental results showing its performance and fairness.

* The definition of beta__cubic_ constant was changed in [RFC8312]. For example, beta__cubic_ in the original paper was the window decrease constant while [RFC8312] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [RFC8312] was beta__cubic_ * _W_max_ while it was (1-beta__cubic_) * _W_max_ in the original paper.

* Its pseudocode used _W_(last_max)_ while [RFC8312] used _W_max_.

* Its AIMD-friendly window was _W_tcp_ while [RFC8312] used _W_est_.

Appendix C. Proof of the Average CUBIC Window Size

This appendix contains a proof for the average CUBIC window size _AVG_W_cubic_ in Figure 6.

We find _AVG_W_cubic_ under a deterministic loss model, where the number of packets between two successive packet losses is 1/_p_. With this model, CUBIC always operates with the concave window profile and the time period between two successive packet losses is _K_.

The average window size _AVG_W_cubic_ is defined as follows, where the numerator 1/_p_ is the total number of packets between two successive packet losses, and the denominator _K_/_RTT_ is the total number of RTTs between two successive packet losses.

$$AVG_W_{cubic} = \frac{1}{\frac{p}{K}} \cdot RTT$$

Figure 9

Below, we find _K_ as a function of CUBIC parameters __cubic_ and _C_, and network parameters _p_ and _RTT_. According to the definition of _K_ in Figure 2, we have

$$K = \sqrt[3]{\frac{W_{max} - W_{max} * _cubic}{C}}$$

Figure 10

The total number of packets between two successive packet losses can also be obtained as follows using the window increase function in Figure 1. Specifically, the window size in the first RTT (i.e., _n_=1 or equivalently _t_=0) is _C_(-_K_)^3+_W_max_ and the window size in the last RTT (i.e., _n_=_K_/_RTT_ or equivalently _t_=_K_-_RTT_) is _C_(-_RTT_)^3+_W_max_.

```
                    K

                   RTT

             1                      3
              =   C((n-1) * RTT-K)  + W
             p                          max
                 n=1
                  K

                 RTT

                           3    3
               =   C * RTT (-n)  + W
                                   max
                 n=1
                               K

                             RTT

                          3       3           K
               = -C * RTT  *    n  + W    *
                                     max   RTT
                             n=1
                                     4
                          3   1   K             K
                -C * RTT  *  *  + W    *
                             4  RTT     max   RTT
                           4
                      1    K            K
               = -C *  *  + W    *
                      4   RTT    max   RTT
```

Figure 11

After solving Figure 10 and Figure 11 for _K_ and _W_max_, we have

```
                          4 * 1-
                     4          cubic    RTT
                 K =    *
                       C * 3 +          p
                                cubic
```

Figure 12

The average CUBIC window size _AVG_W_cubic_ can be obtained by substituting _K_ with Figure 12 in Figure 9

```
                            1
                                   C * 3 +            3
                            p    4           cubic   RTT
              AVG_W      =  =    *
                   cubic    K       4 * 1-           3
                                        cubic     p
                          RTT
```

Authors' Addresses

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
United States of America
Email: xu@unl.edu
URI: https://cse.unl.edu/~xu/

Sangtae Ha
University of Colorado at Boulder
Department of Computer Science
Boulder, CO 80309-0430
United States of America
Email: sangtae.ha@colorado.edu
URI: https://netstech.org/sangtaeha/

Injong Rhee
Bowery Farming
151 W 26TH Street, 12TH Floor
New York, NY 10001
United States of America
Email: injongrhee@gmail.com

Vidhi Goel
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America
Email: vidhi_goel@apple.com

Lars Eggert (editor)
NetApp
Stenbergintie 12 B
FI-02700 Kauniainen
Finland
Email: lars@eggert.org
URI: https://eggert.org/

TCPM WG
Internet Draft
Updates: 793
Intended status: Standards Track
Expires: April 2022

J. Touch
Independent consultant
Wes Eddy
MTI Systems

October 12, 2021

TCP Extended Data Offset Option
draft-ietf-tcpm-tcp-edo-11.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 12, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

TCP segments include a Data Offset field to indicate space for TCP options but the size of the field can limit the space available for complex options such as SACK and Multipath TCP and can limit the combination of such options supported in a single connection. This document updates RFC 793 with an optional TCP extension to that space to support the use of multiple large options. It also explains why the initial SYN of a connection cannot be extending a single segment.

Table of Contents

1. Introduction....3
2. Conventions used in this document....3
3. Motivation....3
4. Requirements for Extending TCP's Data Offset....4
5. The TCP EDO Option....4
   5.1. EDO Supported....5
   5.2. EDO Extension....5
   5.3. The two EDO Extension variants....8
6. TCP EDO Interaction with TCP....9
   6.1. TCP User Interface....9
   6.2. TCP States and Transitions....9
   6.3. TCP Segment Processing....10
   6.4. Impact on TCP Header Size....10
   6.5. Connectionless Resets....11
   6.6. ICMP Handling....11
7. Interactions with Middleboxes....12
   7.1. Middlebox Coexistence with EDO....12
   7.2. Middlebox Interference with EDO....13
8. Comparison to Previous Proposals....14
   8.1. EDO Criteria....14
   8.2. Summary of Approaches....15
   8.3. Extended Segments....16
   8.4. TCPx2....16
   8.5. LO/SLO....17
   8.6. LOIC....17
   8.7. Problems with Extending the Initial SYN....18
9. Implementation Issues....19
10. Security Considerations....20
11. IANA Considerations....20
12. References....20

12.1. Normative References.....20
12.2. Informative References.....20
13. Acknowledgments.....22

1. Introduction

TCP's Data Offset (DO)is a 4-bit field, which indicates the number of 32-bit words of the entire TCP header [RFC793]. This limits the current total header size to 60 bytes, of which the basic header occupies 20, leaving 40 bytes for options. These 40 bytes are increasingly becoming a limitation to the development of advanced capabilities, such as when SACK [RFC2018][RFC6675] is combined with either Multipath TCP [RFC6824], TCP-AO [RFC5925], or TCP Fast Open [RFC7413].

This document specifies the TCP Extended Data Offset (EDO) option, and is independent of (and thus compatible with) IPv4 and IPv6. EDO extends the space available for TCP options, except for the initial SYN and SYN/ACK. This document also explains why the option space of the initial SYN segments cannot be extended as individual segments without severe impact on TCP's initial handshake and the SYN/ACK limitation that results from potential middlebox misbehavior. Multiple other TCP extensions are being considered in the TCPM working group in order to address the case of SYN and SYN/ACK segments [Bo14][Br14][To18]. Some of these other extensions can work in conjunction with EDO (e.g., [To18]).

2. Conventions used in this document

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 RFC-2119 [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.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Motivation

TCP supports headers with a total length of up to 15 32-bit words, as indicated in the 4-bit Data Offset field [RFC793]. This accounts

for a total of 60 bytes, of which the default TCP header fields occupy 20 bytes, leaving 40 bytes for options.

TCP connections already use this option space for a variety of capabilities. These include Maximum Segment Size (MSS) [RFC793], Window Scale (WS) [RFC7323], Timestamp (TS) [RFC7323], Selective Acknowledgement (SACK) [RFC2018][RFC6675], TCP Authentication Option (TCP-AO) [RFC5925], Multipath TCP (MP-TCP)_[RFC6824], and TCP User Timeout [RFC5482]. Some options occur only in a SYN or SYN/ACK (MSS, WS), and others vary in size when used in SYN vs. non-SYN segments.

Each of these options consumes space, where some options consuming as much space as available (SACK) and other desired combinations can easily exceed the currently available space. For example, it is not currently possible to use TCP-AO with both TS and MP-TCP in the same non-SYN segment, i.e., to combine accurate round-trip estimation, authentication, and multipath support in the same connection - even though these options can be negotiated during a SYN exchange (10 for TS, 16 for TCP-AO, and 12 for MP-TCP).

TCP EDO is intended to overcome this limitation for non-SYN segments, as well as to increase the space available for SACK blocks. Further discussion of the impact of EDO and existing options is discussed in Section 6.4. Extending SYN segments is much more complicated, as discussed in Section 8.7.

4. Requirements for Extending TCP’s Data Offset

The primary goal of extending the TCP Data Offset field is to increase the space available for TCP options in all segments except the initial SYN.

An important requirement of any such extension is that it not impact legacy endpoints. Endpoints seeking to use this new option should not incur additional delay or segment exchanges to connect to either new endpoints supporting this option or legacy endpoints without this option. We call this a "backward downgrade" capability.

An additional consideration of this extension is avoiding user data corruption in the presence of popular network devices, including middleboxes. Consideration of middlebox misbehavior can also interfere with extension in the SYN/ACK.

5. The TCP EDO Option

TCP EDO extends the option space for all segments except the initial SYN (i.e., SYN set and ACK not set) and SYN/ACK response. EDO is

indicated by the TCP option codepoint of EDO-OPT and has two types: EDO Supported and EDO Extension, as discussed in the following subsections.

5.1. EDO Supported

EDO capability is determined in both directions using a single exchange of the EDO Supported option (Figure 1). When EDO is desired on a given connection, the SYN and SYN/ACK segments include the EDO Supported option, which consists of the two required TCP option fields: Kind and Length. The EDO Supported option is used only in the SYN and SYN/ACK segments and only to confirm support for EDO in subsequent segments.

```
+--------+--------+
|  Kind  | Length |
+--------+--------+
```

Figure 1 TCP EDO Supported option

An endpoint seeking to enable EDO includes the EDO Supported option in the initial SYN. If receiver of that SYN agrees to use EDO, it responds with the EDO Supported option in the SYN/ACK. The EDO Supported option does not extend the TCP option space.

>> Connections using EDO MUST negotiate its availability during the SYN exchange of the initial three-way handshake.

>> An endpoint confirming and agreeing to EDO use MUST respond with the EDO Supported option in its SYN/ACK.

The SYN/ACK uses only the EDO Supported option (and not the EDO Extension option, below) because it may not yet be safe to extend the option space in the reverse direction due to potential middlebox misbehavior (see Section 7.2). Extension of the SYN and SYN/ACK space is addressed as a separate option (see Section 8.7).

5.2. EDO Extension

When EDO is successfully negotiated, all other segments use the EDO Extension option, of which there are two variants (Figure 2 and Figure 3). Both variants are considered equivalent and either variant can be used in any segment where the EDO Extension option is required. Both variants add a Header_Length field (in network-standard byte order), indicating the length of the entire TCP header in 32-bit words. Figure 3 depicts the longer variant, which includes an additional Segment_Length field, which is identical to the TCP

pseudoheader TCP Length field and used to detect when segments have been altered in ways that would interfere with EDO (discussed further in Section 5.3).

```
+--------+--------+--------+--------+
|  Kind  | Length |  Header_Length  |
+--------+--------+--------+--------+
```

Figure 2 TCP EDO Extension option - simple variant

```
+--------+--------+--------+--------+
|  Kind  | Length |  Header_Length  |
+--------+--------+--------+--------+
|  Segment_Length |
+--------+--------+
```

Figure 3 TCP EDO Extension option - with segment length verification

>> Once enabled on a connection, all segments in both directions MUST include the EDO Extension option. Segments not needing extension MUST set the EDO Extension option Header Length field equal to the Data Offset length.

>> The EDO Extension option MAY be used only if confirmed when the connection transitions to the ESTABLISHED state, e.g., a client is enabled after receiving the EDO Supported option in the SYN/ACK and the server is enabled after seeing the EDO Extension option in the final ACK of the three-way handshake. If either of those segments lacks the appropriate EDO option, the connection MUST NOT use any EDO options on any other segments.

Internet paths may vary after connection establishment, introducing misbehaving middleboxes (see Section 7.2). Using EDO on all segments in both directions allows this condition to be detected.

>> The EDO Supported option MAY occur in an initial SYN as desired (e.g., as expressed by the user/application) and in the SYN/ACK as confirmation, but MUST NOT be inserted in other segments. If the EDO Supported option is received in other segments, it MUST be silently ignored.

>> If EDO has not been negotiated and agreed, the EDO Extension option MUST be silently ignored on subsequent segments. The EDO Extension option MUST NOT be sent in an initial SYN segment or SYN/ACK, and MUST be silently ignored and not acknowledged if so received.

>> If EDO has been negotiated, any subsequent segments arriving without the EDO Extension option MUST be silently ignored. Such events MAY be logged as warning errors and logging MUST be rate limited.

When processing a segment, EDO needs to be visible within the area indicated by the Data Offset field, so that processing can use the EDO Header_length to override the field for that segment.

>> The EDO Extension option MUST occur within the space indicated by the TCP Data Offset.

>> The EDO Extension option indicates the total length of the header. The EDO Header_length field MUST NOT exceed that of the total segment size (i.e., TCP Length).

>> The EDO Header Length MUST be at least as large as the TCP Data Offset field of the segment in which they both appear. When the EDO Header Length equals the Data Offset length, the EDO Extension option is present but it does not extend the option space. When the EDO Header Length is invalid, the TCP segment MUST be silently dropped.

>> The EDO Supported option SHOULD be aligned on a 16-bit boundary and the EDO Extension option SHOULD be aligned on a 32-bit boundary, in both cases for simpler processing.

For example, a segment with only EDO would have a Data Offset of 6 or 7 (depending on the EDO Extension variant used), where EDO would be the first option processed, at which point the EDO Extension option would override the Data Offset and processing would continue until the end of the TCP header as indicated by the EDO Header_length field.

There are cases where it might be useful to process other options before EDO, notably those that determine whether the TCP header is valid, such as authentication, encryption, or alternate checksums. In those cases, the EDO Extension option is preferably the first option after a validation option, and the payload after the Data Offset is treated as user data for the purposes of validation.

>> The EDO Extension option SHOULD occur as early as possible, either first or just after any authentication or encryption, and SHOULD be the last option covered by the Data Offset value.

Other options are generally handled in the same manner as when the EDO option is not active, unless they interact with other options.

One such example is TCP-AO [RFC5925], which optionally ignores the contents of TCP options, so it would need to be aware of EDO to operate correctly when options are excluded from the HMAC calculation.

>> Options that depend on other options, such as TCP-AO [RFC5925] (which may include or exclude options in MAC calculations) MUST also be augmented to interpret the EDO Extension option to operate correctly.

5.3. The two EDO Extension variants

There are two variants of the EDO Extension option; one includes a copy of the TCP segment length, copied from the TCP pseduoheader [RFC793]. The Segment_Length field is added to the longer variant to detect when segments are incorrectly and inappropriately merged by middleboxes or TCP offload processing but without consideration for the additional option space indicated by the EDO Header_Length field. Such effects are described in further detail in Section 7.2.

>> An endpoint MAY use either variant of the EDO Extension option interchangeably.

When the longer, 6-byte variant is used, the Segment_Length field is used to check whether modification of the segment was performed consistent with knowledge of the EDO option. The Segment_Length field will detect any modification of the length of the segment, such as might occur when segments are split or merged, that occurs without also updating the Segment Length field as well. The Segment Length field thus helps endpoints detects devices that merge or split TCP segments without support for EDO. Devices that merge or split TCP segments that support EDO would update the Segment Length field as needed but would also ensure that the user data is handled separately from the extended option space indicate by EDO.

>> When an endpoint creates a new segment using the 6-byte EDO Extension option, the Segment_Length field is initialized with a copy of the segment length from the TCP pseudoheader.

>> When an endpoint receives a segment using the 6-byte EDO Extension option, it MUST validate the Segment_Length field with the length of the segment as indicated in the TCP pseudoheader. If the segment lengths do not match, the segment MUST be discarded and an error SHOULD be logged in a rate-limited manner.

>> The 6-byte EDO Extension variant SHOULD be used where middlebox or TCP offload support could merge or split TCP segments without

consideration for the EDO option. Because these conditions could occur at either endpoint or along the network path, the 6-byte variant SHOULD be preferred until sufficient evidence for safe use of the 4-byte variant is determined by the community.

The field will not detect other modification of the TCP user data; such modifications would need more complex detection mechanisms, such as checksums or hashes. When these are used, as with IPsec or TCP-AO, the 4-byte variant is sufficient.

>> The 4-byte EDO Extension variant is sufficient when EDO is used in conjunction with other mechanisms that provide integrity protection, such as IPsec or TCP-AO.

6. TCP EDO Interaction with TCP

The following subsections describe how EDO interacts with the TCP specification [RFC793].

6.1. TCP User Interface

The TCP EDO option is enabled on a connection using a mechanism similar to any other per-connection option. In Unix systems, this is typically performed using the 'setsockopt' system call.

>> Implementations can also employ system-wide defaults, however systems SHOULD NOT activate this extension by default to avoid interfering with legacy applications.

>> Due to the potential impacts of legacy middleboxes (discussed in Section 7), a TCP implementation supporting EDO SHOULD log any events within an EDO connection when options that are malformed or show other evidence of tampering arrive. An operating system MAY choose to cache the list of destination endpoints where this has occurred with and block use of EDO on future connections to those endpoints, but this cache MUST be accessible to users/applications on the host. Note that such endpoint assumptions can vary in the presence of load balancers where server implementations vary behind such balancers.

6.2. TCP States and Transitions

TCP EDO does not alter the existing TCP state or state transition mechanisms.

6.3. TCP Segment Processing

TCP EDO alters segment processing during the TCP option processing step. Once detected, the TCP EDO Extension option overrides the TCP Data Offset field for all subsequent option processing. Option processing continues at the next option (if present) after the EDO Extension option.

6.4. Impact on TCP Header Size

The TCP EDO Supported option increases SYN header length by a minimum of 2 bytes but could increase it by more depending on 32-bit word alignment. Currently popular SYN options total 19 bytes, which leaves more than enough room for the EDO Supported option:

- o SACK permitted (2 bytes in SYN, optionally 2 + 8N bytes after) [RFC2018][RFC6675]

- o Timestamp (10 bytes) [RFC7323]

- o Window scale (3 bytes) [RFC7323]

- o MSS option (4 bytes) [RFC793]

Adding the EDO Supported option would result in a total of 21 bytes of SYN option space.

Subsequent segments would use 10 bytes of option space without any SACK blocks (TS only; WS and MSS are used only in SYN and SYN/ACK) or allow up to 3 SACK blocks before needing to use EDO; with EDO, the number of SACK blocks or additional options would be substantially increased. There are also other options that are emerging in the SYN, including TCP Fast Open, which uses another 6-18 (typically 10) bytes in the SYN/ACK of the first connection and in the SYN of subsequent connections [RFC7413].

TCP EDO can also be negotiated in SYNs with either of the following large options:

- o TCP-AO (authentication) (16 bytes) [RFC5925]

- o Multipath TCP (12 bytes in SYN and SYN/ACK, 20 after) [RFC6824]

Including TCP-AO with TS, WS, SACK increases the SYN option space use to 35 bytes; with Multipath TCP the use is 31 bytes. When Multipath TCP is enabled with the typical options, later segments would require 30 bytes without SACK, thus limiting the SACK option

to one block unless EDO is also supported on at least non-SYN segments.

The full combination of the above options (47 bytes for TS, WS, MSS, SACK, TCP-AO, and MPTCP) does not fit in the existing SYN option space and (as noted) that space cannot be extended within a single SYN segment. There has been a proposal to change TS to a 2 byte "TS permitted" signal in the initial SYN, provided it can be safely enabled during the connection later or might be avoided completely [Ni15]. Even using "TS-permitted", the total space is still too large to support in the initial SYN without SYN option space extension [Bo14][Br14][To18].

The EDO Extension option has negligible impact on other headers because it can either come first or just after security information, and in either case the additional 4 or 6 bytes are easily accommodated within the TCP Data Offset length. Once the EDO option is processed, the entirety of the remainder of the TCP segment is available for any remaining options.

6.5. Connectionless Resets

A RST may arrive during a currently active connection or may be needed to cleanup old state from an abandoned connection. The latter occurs when a new SYN is sent to an endpoint with matching existing connection state, at which point that endpoint responds with a RST and both ends remove stale information.

The EDO Extension option is mandatory on all TCP segments once negotiated, i.e., except in the SYN and SYN/ACK (which establish support) and the RST. A RST may lack the context to know that EDO is active on a connection.

>> The EDO Extension option MAY occur in a RST when the endpoint has connection state that has negotiated EDO. However, unless the RST is generated by an incoming segment that includes an EDO Extension option, the transmitted RST MUST NOT include the EDO Extension option.

6.6. ICMP Handling

ICMP responses are intended to include the IP and the port fields of TCP and UDP headers of typical TCP/IP and UDP/IP packets [RFC792]. This includes the first 8 data bytes of the original datagram, intended to include the transport port numbers used for connection demultiplexing. Later specifications encourage returning as much of the original payload as possible [RFC1812]. In either case, legacy

options or new options in the EDO extension area might or might not be included, and so options are generally not assumed to be part of ICMP processing anyway.

7. Interactions with Middleboxes

Middleboxes are on-path devices that typically examine or modify packets in ways that Internet routers do not [RFC3234]. This includes parsing transport headers and/or rewriting transport segments in ways that may affect EDO.

There are several cases to consider:

- Typical NAT/NAPT devices, which modify only IP address and/or TCP port number fields (with associated TCP checksum updates)

- Middleboxes that try to reconstitute TCP data streams, such as for deep-packet inspection for virus scanning

- Middleboxes that modify known TCP header fields

- Middleboxes that rewrite TCP segments

7.1. Middlebox Coexistence with EDO

Middleboxes can coexist with EDO when they either support EDO or when they ignore its impact on segment structure.

NATs and NAPTs, which rewrite IP address and/or transport port fields, are the most common form of middlebox and are not affected by the EDO option.

Middleboxes that support EDO would be those that correctly parse the EDO option. Such boxes can reconstitute the TCP data stream correctly or can modify header fields and/or rewrite segments without impact to EDO.

Conventional TCP proxies terminate the TCP connection in both directions and thus operate as TCP endpoints, such as when a client-middlebox and middlebox-server each have separate TCP connections. They would support EDO by following the host requirements herein on both connections. The use of EDO on one connection is independent of its use on the other in this case.

7.2. Middlebox Interference with EDO

Middleboxes that do not support EDO cannot coexist with its use when they modify segment boundaries or do not forward unknown (e.g., the EDO) options.

So-called "transparent" rewriting proxies, which inappropriately and incorrectly modify TCP segment boundaries, might mix option information with user data if they did not support EDO. Such devices might also interfere with other TCP options such as TCP-AO. There are three types of such boxes:

- Those that process received options and transmit sent options separately, i.e., although they rewrite segments, they behave as TCP endpoints in both directions.

- Those that split segments, taking a received segment and emitting two or more segments with revised headers.

- Those that join segments, receiving multiple segments and emitting a single segment whose data is the concatenation of the components.

In all three cases, EDO is either treated as independent on different sides of such boxes or not. If independent, EDO would either be correctly terminated in either or both directions or disabled due to lack of SYN/ACK confirmation in either or both directions. Problems would occur only when TCP segments with EDO are combined or split while ignoring the EDO option. In the split case, the key concern is if the split happens within the option extension space or if EDO is silently copied to both segments without copying the corresponding extended option space contents. However, the most comprehensive study of these cases indicates that "although middleboxes do split and coalesce segments, none did so while passing unknown options" [Ho11].

Note that the second and third types of middlebox behaviors listed above may create syndromes similar to TCP transmit and receive hardware offload engines that incorrectly modify segments with unknown options.

Middleboxes that silently remove options that they do not implement have been observed [Ho11]. Such boxes interfere with the use of the EDO Extension option in the SYN and SYN/ACK segments because extended option space would be misinterpreted as user data if the EDO Extension option were removed, and this cannot be avoided. This is one reason that SYN and SYN/ACK extension requires alternate

mechanisms (see Section 8.7). It is also the reason for the 6-byte EDO Extension variant (see Section 5.3), which can detect such merging or splitting of segments. Further, if such middleboxes become present on a path they could cause similar misinterpretation on segments exchanged in the ESTABLISHED and subsequent states. As a result, this document requires that the EDO Extension option be avoided on the SYN/ACK and that this option needs to be used on all segments once successfully negotiated and encourages use of the 6-byte EDO Extension variant.

Deep-packet inspection systems that inspect TCP segment payloads or attempt to reconstitute the data stream would incorrectly include option data in the reconstituted user data stream, which might interfere with their operation.

>> It can be important to detect misbehavior that could cause EDO space to be misinterpreted as user data. In such cases, EDO SHOULD be used in conjunction with an integrity protection mechanism. This includes the 6-byte EDO Extension variant or stronger mechanisms such as IPsec, TCP-AO, etc. It is useful to note that such protection only helps non-compliant components and enable avoidance (e.g., disabling EDO), but integrity protection alone cannot correct the misinterpretation of EDO space as user data.

This situation is similar to that of ECN and ICMP support in the Internet. In both cases, endpoints have evolved mechanisms for detecting and robustly operating around "black holes". Very similar algorithms are expected to be applicable for EDO.

8. Comparison to Previous Proposals

EDO is the latest in a long line of attempts to increase TCP option space [Al06][Ed08][Ko04][Ra12][Yo11]. The following is a comparison of these approaches to EDO, based partly on a previous summary [Ra12]. This comparison differs from that summary by using a different set of success criteria.

8.1. EDO Criteria

Our criteria for a successful solution are as follows:

- o Zero-cost fallback to legacy endpoints.
- o Minimal impact on middlebox compatibility.
- o No additional side-effects.

Zero-cost fallback requires that upgraded hosts incur no penalty for attempting to use EDO. This disqualifies dual-stack approaches, because the client might have to delay connection establishment to wait for the preferred connection mode to complete. Note that the impact of legacy endpoints that silently reflect unknown options are not considered, as they are already non-compliant with existing TCP requirements [RFC793].

Minimal impact on middlebox compatibility requires that EDO works through simple NAT and NAPT boxes, which modify IP addresses and ports and recompute IPv4 header and TCP segment checksums. Middleboxes that reject unknown options or that process segments in detail without regard for unknown options are not considered; they process segments as if they were an endpoint but do so in ways that are not compliant with existing TCP requirements (e.g., they should have rejected the initial SYN because of its unknown options rather than silently relaying it).

EDO also attempts to avoid creating side-effects, such as might happen if options were split across multiple TCP segments (which could arrive out of order or be lost) or across different TCP connections (which could fail to share fate through firewalls or NAT/NAPTs).

These requirements are similar to those noted in [Ra12], but EDO groups cases of segment modification beyond address and port - such as rewriting, segment drop, sequence number modification, and option stripping - as already in violation of existing TCP requirements regarding unknown options, and so we do not consider their impact on this new option.

8.2. Summary of Approaches

There are three basic ways in which TCP option space extension has been attempted:

1. Use of a TCP option.

2. Redefinition of the existing TCP header fields.

3. Use of option space in multiple TCP segments (split across multiple segments).

A TCP option is the most direct way to extend the option space and is the basis of EDO. This approach cannot extend the option space of the initial SYN.

Redefining existing TCP header fields can be used to either contain additional options or as a pointer indicating alternate ways to interpret the segment payload. All such redefinitions make it difficult to achieve zero-impact backward compatibility, both with legacy endpoints and middleboxes.

Splitting option space across separate segments can create unintended side-effects, such as increased delay to deal with path latency or loss differences.

The following discusses three of the most notable past attempts to extend the TCP option space: Extended Segments, TCPx2, LO/SLO, and LOIC. [Ra12] suggests a few other approaches, including use of TCP option cookies, reuse/overload of other TCP fields (e.g., the URG pointer), or compressing TCP options. None of these is compatible with legacy endpoints or middleboxes.

8.3. Extended Segments

TCP Extended Segments redefined the meaning of currently unused values of the Data Offset (DO) field [Ko04]. TCP defines DO as indicating the length of the TCP header, including options, in 32-bit words. The default TCP header with no options is 5 such words, so the minimum currently valid DO value is 5 (meaning 40 bytes of option space). This document defines interpretations of values 0-4: DO=0 means 48 bytes of option space, DO=1 means 64, DO=2 means 128, DO=3 means 256, and DO=4 means unlimited (e.g., the entire payload is option space). This variant negotiates the use of this capability by using one of these invalid DO values in the initial SYN.

Use of this variant is not backward-compatible with legacy TCP implementations, whether at the desired endpoint or on middleboxes. The variant also defines a way to initiate the feature on the passive side, e.g., using an invalid DO during the SYN/ACK when the initial SYN had a valid DO. This capability allows either side to initiate use of the feature but is also not backward compatible.

8.4. TCPx2

TCPx2 redefines legacy TCP headers by basically doubling all TCP header fields [Al06]. It relies on a new transport protocol number to indicate its use, defeating backward compatibility with all existing TCP capabilities, including firewalls, NATs/NAPTs, and legacy endpoints and applications.

8.5. LO/SLO

The TCP Long Option (LO, [Ed08]) is very similar to EDO, except that presence of LO results in ignoring the existing Data Offset (DO) field and that LO is required to be the first option. EDO considers the need for other fields to be first and declares that the EDO is the last option as indicated by the DO field value. Like LO, EDO is required in every segment once negotiated.

The TCP Long Option draft also specified the SYN Long Option (SLO) [Ed08]. If SLO is used in the initial SYN and successfully negotiated, it is used in each subsequent segment until all of the initial SYN options are transmitted.

LO is backward compatible, as is SLO; in both cases, endpoints not supporting the option would not respond with the option, and in both cases the initial SYN is not itself extended.

SLO does modify the three-way handshake because the connection isn't considered completely established until the first data byte is acknowledged. Legacy TCP can establish a connection even in the absence of data. SLO also changes the semantics of the SYN/ACK; for legacy TCP, this completes the active side connection establishment, where in SLO an additional data ACK is required. A connection whose initial SYN options have been confirmed in the SYN/ACK might still fail upon receipt of additional options sent in later SLO segments. This case - of late negotiation fail - is not addressed in the specification.

8.6. LOIC

TCP Long Options by Invalid Checksum is a dual-stack approach that uses two initial SYNS to initiate all updated connections [Yo11]. One SYN negotiates the new option and the other SYN payload contains only the entire options. The negotiation SYN is compliant with existing procedures, but the option SYN has a deliberately incorrect TCP checksum (decremented by 2). A legacy endpoint would discard the segment with the incorrect checksum and respond to the negotiation SYN without the LO option.

Use of the option SYN and its incorrect checksum both interfere with other legacy components. Segments with incorrect checksums will be silently dropped by most middleboxes, including NATs/NAPTs. Use of two SYNs creates side-effects that can delay connections to upgraded endpoints, notably when the option SYN is lost or the SYNs arrive out of order. Finally, by not allowing other options in the negotiation SYN, all connections to legacy endpoints either use no

options or require a separate connection attempt (either concurrent or subsequent).

8.7. Problems with Extending the Initial SYN

The key difficulty with most previous proposals is the desire to extend the option space in all TCP segments, including the initial SYN, i.e., SYN with no ACK, typically the first segment of a connection, as well as possibly the SYN/ACK. It has proven difficult to extend space within the segment of the initial SYN in the absence of prior negotiation while maintaining current TCP three-way handshake properties, and it may be similarly challenging to extend the SYN/ACK (depending on asymmetric middlebox assumptions).

A new TCP option cannot extend the Data Offset of a single TCP initial SYN segment and cannot extend a SYN/ACK in a single segment when considering misbehaving middleboxes. All TCP segments, including the initial SYN and SYN/ACK, may include user data in the payload data [RFC793], and this can be useful for some proposed features such as TCP Fast Open [RFC7413]. Legacy endpoints that ignore the new option would process the payload contents as user data and send an ACK. Once ACK'd, this data cannot be removed from the user stream.

The Reserved TCP header bits cannot be redefined easily, even though three of the six total bits have already been redefined (ECE/CWR [RFC3168] and NS [RFC3540]). Legacy endpoints have been known to reflect received values in these fields; this was safely dealt with for ECN but would be difficult here [RFC3168].

TCP initial SYN (SYN and not ACK) segments can use every other TCP header field except the Acknowledgement number, which is not used because the ACK field is not set. In all other segments, all fields except the three remaining Reserved header bits are actively used. The total amount of available header fields, in either case, is insufficient to be useful in extending the option space.

The representation of TCP options can be optimized to minimize the space needed. In such cases, multiple Kind and Length fields are combined, so that a new Kind would indicate a specific combination of options, whose order is fixed and whose length is indicated by one Length field. Most TCP options use fields whose size is much larger than the required Kind and Length components, so the resulting efficiency is typically insufficient for additional options.

The option space of an initial SYN segment might be extended by using multiple initial segments (e.g., multiple SYNs or a SYN and non-SYN) or based on the context of previous or parallel connections. This method may also be needed to extend space in the SYN/ACK in the presence of misbehaving middleboxes. Because of their potential complexity, these approaches are addressed in separate documents [Bo14][Br14][To18].

Option space cannot be extended in outer layer headers, e.g., IPv4 or IPv6. These layers typically try to avoid extensions altogether, to simplify forwarding processing at routers. Introducing new shim layers to accommodate additional option space would interfere with deep-packet inspection mechanisms that are in widespread use.

As a result, EDO does not attempt to extend the space available for options in TCP initial SYNs. It does extend that space in all other segments (including SYN/ACK), which has always been trivially possible once an option is defined.

9. Implementation Issues

TCP segment processing can involve accessing nonlinear data structures, such as chains of buffers. Such chains are often designed so that the maximum default TCP header (60 bytes) fits in the first buffer. Extending the TCP header across multiple buffers may necessitate buffer traversal functions that span boundaries between buffers. Such traversal can also have a significant performance impact, which is additional rationale for using TCP option space - even extended option space - sparingly.

Although EDO can be large enough to consume the entire segment, it is important to leave space for data so that the TCP connection can make forward progress. It would be wise to limit EDO to consuming no more than MSS-4 bytes of the IP segment, preferably even less (e.g., MSS-128 bytes).

When using the ExID variant for testing and experimentation, either TCP option codepoint (253, 254) is valid in sent or received segments.

Implementers need to be careful about the potential for offload support interfering with this option. The EDO data needs to be passed to the protocol stack as part of the option space, not integrated with the user segment, to allow the offload to independently determine user data segment boundaries and combine them correctly with the extended option data. Some legacy hardware receive offload engines may present challenges in this regard, and

may be incompatible with EDO where they incorrectly attempt to process segments with unknown options. Such offload engines are part of the protocol stack and updated accordingly. Issues with incorrect resegmentation by an offload engine can be detected in the same way as middlebox tampering.

10. Security Considerations

It is meaningless to have the Data Offset further exceed the position of the EDO data offset option.

>> When the EDO Extension option is present, the EDO Extension option SHOULD be the last non-null option covered by the TCP Data Offset, because it would be the last option affected by Data Offset.

This also makes it more difficult to use the Data Offset field as a covert channel.

11. IANA Considerations

We request that, upon publication, this option be assigned a TCP Option codepoint by IANA, which the RFC Editor will replace EDO-OPT in this document with codepoint value.

The TCP Experimental ID (ExID) with a 16-bit value of 0x0ED0 (in network standard byte order) has been assigned for use during testing and preliminary experiments.

12. References

12.1. Normative References

[RFC793] 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.

12.2. Informative References

[Al06] Allman, M., "TCPx2: Don’t Fence Me In", draft-allman-tcpx2-hack-00 (work in progress), May 2006.

[Bo14] Borman, D., "TCP Four-Way Handshake", draft-borman-tcp4way-00 (work in progress), October 2014.

[Br14] Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[Ho11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP", Proc. ACM Sigcomm Internet Measurement Conference (IMC), 2011, pp. 181-194.

[Ko04] Kohler, E., "Extended Option Space for TCP", draft-kohler-tcpm-extopt-00 (work in progress), September 2004.

[Ni15] Nishida, Y., "A-PAWS: Alternative Approach for PAWS", draft-nishida-tcpm-apaws-02 (work in progress), Oct. 2015.

[Ra12] Ramaiah, A., "TCP option space extension", draft-ananth-tcpm-tcpoptext-00 (work in progress), March 2012.

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792, September 1981.

[RFC1812] Baker, F. (Ed.), "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

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

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

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 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 (Ed.), "TCP Extensions for High Performance", RFC 7323, September 2014.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

[To21] Touch, J., T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt (work in progress), Oct. 2019.

[Yo11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

13. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback, in particular: Oliver Bonaventure, Bob Briscoe, Ted Faber, John Leslie, Pasi Sarolahti, Richard Scheffenegger, and Alexander Zimmerman.

This work is partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors’ Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Wesley M. Eddy
MTI Systems
US

Email: wes@mti-systems.com

TCPM WG
Internet Draft
Updates: 9293
Intended status: Standards Track
Expires: April 2023

J. Touch
Independent Consultant
Wes Eddy
MTI Systems

October 22, 2022

TCP Extended Data Offset Option
draft-ietf-tcpm-tcp-edo-13.txt

Abstract

TCP segments include a Data Offset field to indicate space for TCP options but the size of the field can limit the space available for complex options such as SACK and Multipath TCP and can limit the combination of such options supported in a single connection. This document updates RFC 9293 with an optional TCP extension to that space to support the use of multiple large options. It also explains why the initial SYN of a connection cannot be extending a single segment.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

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 22, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction..... 3
2. Conventions used in this document..... 3
3. Motivation..... 3
4. Requirements for Extending TCP's Data Offset..... 4
5. The TCP EDO Option..... 4
   5.1. EDO Supported..... 5
   5.2. EDO Extension..... 5
   5.3. The two EDO Extension variants..... 8
6. TCP EDO Interaction with TCP..... 9
   6.1. TCP User Interface..... 9
   6.2. TCP States and Transitions..... 9
   6.3. TCP Segment Processing..... 10
   6.4. Impact on TCP Header Size..... 10
   6.5. Connectionless Resets..... 11
   6.6. ICMP Handling..... 11
7. Interactions with Middleboxes..... 12
   7.1. Middlebox Coexistence with EDO..... 12
   7.2. Middlebox Interference with EDO..... 13
8. Comparison to Previous Proposals..... 14
   8.1. EDO Criteria..... 14
   8.2. Summary of Approaches..... 15
   8.3. Extended Segments..... 16
   8.4. TCPx2..... 16
   8.5. LO/SLO..... 17
   8.6. LOIC..... 17
   8.7. Problems with Extending the Initial SYN..... 18
9. Implementation Issues..... 19
10. Security Considerations..... 20
11. IANA Considerations..... 20

12. References..... 20
12.1. Normative References..... 20
12.2. Informative References..... 20
13. Acknowledgments..... 22

1. Introduction

TCP's Data Offset (DO)is a 4-bit field, which indicates the number of 32-bit words of the entire TCP header [RFC9293]. This limits the current total header size to 60 bytes, of which the basic header occupies 20, leaving 40 bytes for options. These 40 bytes are increasingly becoming a limitation to the development of advanced capabilities, such as when SACK [RFC2018][RFC6675] is combined with either Multipath TCP [RFC8684], TCP-AO [RFC5925], or TCP Fast Open [RFC7413].

This document specifies the TCP Extended Data Offset (EDO) option, and is independent of (and thus compatible with) IPv4 and IPv6. EDO extends the space available for TCP options, except for the initial SYN and SYN/ACK. This document also explains why the option space of the initial SYN segments cannot be extended as individual segments without severe impact on TCP's initial handshake and the SYN/ACK limitation that results from potential middlebox misbehavior. Multiple other TCP extensions are being considered in the TCPM working group in order to address the case of SYN and SYN/ACK segments [Bo14][Br14][To22]. Some of these other extensions can work in conjunction with EDO (e.g., [To22]).

2. Conventions used in this document

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 RFC-2119 [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.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Motivation

TCP supports headers with a total length of up to 15 32-bit words, as indicated in the 4-bit Data Offset field [RFC9293]. This accounts

for a total of 60 bytes, of which the default TCP header fields occupy 20 bytes, leaving 40 bytes for options.

TCP connections already use this option space for a variety of capabilities. These include Maximum Segment Size (MSS) [RFC9293], Window Scale (WS) [RFC7323], Timestamp (TS) [RFC7323], Selective Acknowledgement (SACK) [RFC2018][RFC6675], TCP Authentication Option (TCP-AO) [RFC5925], Multipath TCP (MP-TCP)_[RFC8684], and TCP User Timeout [RFC5482]. Some options occur only in a SYN or SYN/ACK (MSS, WS), and others vary in size when used in SYN vs. non-SYN segments.

Each of these options consumes space, where some options consuming as much space as available (SACK) and other desired combinations can easily exceed the currently available space. For example, it is not currently possible to use TCP-AO with both TS and MP-TCP in the same non-SYN segment, i.e., to combine accurate round-trip estimation, authentication, and multipath support in the same connection - even though these options can be negotiated during a SYN exchange (10 for TS, 16 for TCP-AO, and 12 for MP-TCP).

TCP EDO is intended to overcome this limitation for non-SYN segments, as well as to increase the space available for SACK blocks. Further discussion of the impact of EDO and existing options is discussed in Section 6.4. Extending SYN segments is much more complicated, as discussed in Section 8.7.

4. Requirements for Extending TCP’s Data Offset

The primary goal of extending the TCP Data Offset field is to increase the space available for TCP options in all segments except the initial SYN.

An important requirement of any such extension is that it not impact legacy endpoints. Endpoints seeking to use this new option should not incur additional delay or segment exchanges to connect to either new endpoints supporting this option or legacy endpoints without this option. We call this a "backward downgrade" capability.

An additional consideration of this extension is avoiding user data corruption in the presence of popular network devices, including middleboxes. Consideration of middlebox misbehavior can also interfere with extension in the SYN/ACK.

5. The TCP EDO Option

TCP EDO extends the option space for all segments except the initial SYN (i.e., SYN set and ACK not set) and SYN/ACK response. EDO is

indicated by the TCP option codepoint of EDO-OPT and has two types: EDO Supported and EDO Extension, as discussed in the following subsections.

5.1. EDO Supported

EDO capability is determined in both directions using a single exchange of the EDO Supported option (Figure 1). When EDO is desired on a given connection, the SYN and SYN/ACK segments include the EDO Supported option, which consists of the two required TCP option fields: Kind and Length. The EDO Supported option is used only in the SYN and SYN/ACK segments and only to confirm support for EDO in subsequent segments.

```
+--------+--------+
|  Kind  | Length |
+--------+--------+
```

Figure 1 TCP EDO Supported option

An endpoint seeking to enable EDO includes the EDO Supported option in the initial SYN. If receiver of that SYN agrees to use EDO, it responds with the EDO Supported option in the SYN/ACK. The EDO Supported option does not extend the TCP option space.

>> Connections using EDO MUST negotiate its availability during the SYN exchange of the initial three-way handshake.

>> An endpoint confirming and agreeing to EDO use MUST respond with the EDO Supported option in its SYN/ACK.

The SYN/ACK uses only the EDO Supported option (and not the EDO Extension option, below) because it may not yet be safe to extend the option space in the reverse direction due to potential middlebox misbehavior (see Section 7.2). Extension of the SYN and SYN/ACK space is addressed as a separate option (see Section 8.7).

5.2. EDO Extension

When EDO is successfully negotiated, all other segments use the EDO Extension option, of which there are two variants (Figure 2 and Figure 3). Both variants are considered equivalent and either variant can be used in any segment where the EDO Extension option is required. Both variants add a Header_Length field (in network-standard byte order), indicating the length of the entire TCP header in 32-bit words. Figure 3 depicts the longer variant, which includes an additional Segment_Length field, which is identical to the TCP

pseudoheader TCP Length field and used to detect when segments have been altered in ways that would interfere with EDO (discussed further in Section 5.3).

```
+--------+--------+--------+--------+
|  Kind  | Length |  Header_Length  |
+--------+--------+--------+--------+
```

Figure 2 TCP EDO Extension option - simple variant

```
+--------+--------+--------+--------+
|  Kind  | Length |  Header_Length  |
+--------+--------+--------+--------+
|  Segment_Length |
+--------+--------+
```

Figure 3 TCP EDO Extension option - with segment length verification

>> Once enabled on a connection, all segments in both directions MUST include the EDO Extension option. Segments not needing extension MUST set the EDO Extension option Header Length field equal to the Data Offset length.

>> The EDO Extension option MAY be used only if confirmed when the connection transitions to the ESTABLISHED state, e.g., a client is enabled after receiving the EDO Supported option in the SYN/ACK and the server is enabled after seeing the EDO Extension option in the final ACK of the three-way handshake. If either of those segments lacks the appropriate EDO option, the connection MUST NOT use any EDO options on any other segments.

Internet paths may vary after connection establishment, introducing misbehaving middleboxes (see Section 7.2). Using EDO on all segments in both directions allows this condition to be detected.

>> The EDO Supported option MAY occur in an initial SYN as desired (e.g., as expressed by the user/application) and in the SYN/ACK as confirmation, but MUST NOT be inserted in other segments. If the EDO Supported option is received in other segments, it MUST be silently ignored.

>> If EDO has not been negotiated and agreed, the EDO Extension option MUST be silently ignored on subsequent segments. The EDO Extension option MUST NOT be sent in an initial SYN segment or SYN/ACK, and MUST be silently ignored and not acknowledged if so received.

>> If EDO has been negotiated, any subsequent segments arriving without the EDO Extension option MUST be silently ignored. Such events MAY be logged as warning errors and logging MUST be rate limited.

When processing a segment, EDO needs to be visible within the area indicated by the Data Offset field, so that processing can use the EDO Header_length to override the field for that segment.

>> The EDO Extension option MUST occur within the space indicated by the TCP Data Offset.

>> The EDO Extension option indicates the total length of the header. The EDO Header_length field MUST NOT exceed that of the total segment size (i.e., TCP Length).

>> The EDO Header Length MUST be at least as large as the TCP Data Offset field of the segment in which they both appear. When the EDO Header Length equals the Data Offset length, the EDO Extension option is present but it does not extend the option space. When the EDO Header Length is invalid, the TCP segment MUST be silently dropped.

>> The EDO Supported option SHOULD be aligned on a 16-bit boundary and the EDO Extension option SHOULD be aligned on a 32-bit boundary, in both cases for simpler processing.

For example, a segment with only EDO would have a Data Offset of 6 or 7 (depending on the EDO Extension variant used), where EDO would be the first option processed, at which point the EDO Extension option would override the Data Offset and processing would continue until the end of the TCP header as indicated by the EDO Header_length field.

There are cases where it might be useful to process other options before EDO, notably those that determine whether the TCP header is valid, such as authentication, encryption, or alternate checksums. In those cases, the EDO Extension option is preferably the first option after a validation option, and the payload after the Data Offset is treated as user data for the purposes of validation.

>> The EDO Extension option SHOULD occur as early as possible, either first or just after any authentication or encryption, and SHOULD be the last option covered by the Data Offset value.

Other options are generally handled in the same manner as when the EDO option is not active, unless they interact with other options.

One such example is TCP-AO [RFC5925], which optionally ignores the contents of TCP options, so it would need to be aware of EDO to operate correctly when options are excluded from the HMAC calculation.

>> Options that depend on other options, such as TCP-AO [RFC5925] (which may include or exclude options in MAC calculations) MUST also be augmented to interpret the EDO Extension option to operate correctly.

5.3. The two EDO Extension variants

There are two variants of the EDO Extension option; one includes a copy of the TCP segment length, copied from the TCP pseduoheader [RFC9293]. The Segment_Length field is added to the longer variant to detect when segments are incorrectly and inappropriately merged by middleboxes or TCP offload processing but without consideration for the additional option space indicated by the EDO Header_Length field. Such effects are described in further detail in Section 7.2.

>> An endpoint MAY use either variant of the EDO Extension option interchangeably.

When the longer, 6-byte variant is used, the Segment_Length field is used to check whether modification of the segment was performed consistent with knowledge of the EDO option. The Segment_Length field will detect any modification of the length of the segment, such as might occur when segments are split or merged, that occurs without also updating the Segment Length field as well. The Segment Length field thus helps endpoints detects devices that merge or split TCP segments without support for EDO. Devices that merge or split TCP segments that support EDO would update the Segment Length field as needed but would also ensure that the user data is handled separately from the extended option space indicate by EDO.

>> When an endpoint creates a new segment using the 6-byte EDO Extension option, the Segment_Length field is initialized with a copy of the segment length from the TCP pseudoheader.

>> When an endpoint receives a segment using the 6-byte EDO Extension option, it MUST validate the Segment_Length field with the length of the segment as indicated in the TCP pseudoheader. If the segment lengths do not match, the segment MUST be discarded and an error SHOULD be logged in a rate-limited manner.

>> The 6-byte EDO Extension variant SHOULD be used where middlebox or TCP offload support could merge or split TCP segments without

consideration for the EDO option. Because these conditions could occur at either endpoint or along the network path, the 6-byte variant SHOULD be preferred until sufficient evidence for safe use of the 4-byte variant is determined by the community.

The field will not detect other modification of the TCP user data; such modifications would need more complex detection mechanisms, such as checksums or hashes. When these are used, as with IPsec or TCP-AO, the 4-byte variant is sufficient.

>> The 4-byte EDO Extension variant is sufficient when EDO is used in conjunction with other mechanisms that provide integrity protection, such as IPsec or TCP-AO.

6. TCP EDO Interaction with TCP

The following subsections describe how EDO interacts with the TCP specification [RFC9293].

6.1. TCP User Interface

The TCP EDO option is enabled on a connection using a mechanism similar to any other per-connection option. In Unix systems, this is typically performed using the 'setsockopt' system call.

>> Implementations can also employ system-wide defaults, however systems SHOULD NOT activate this extension by default to avoid interfering with legacy applications.

>> Due to the potential impacts of legacy middleboxes (discussed in Section 7), a TCP implementation supporting EDO SHOULD log any events within an EDO connection when options that are malformed or show other evidence of tampering arrive. An operating system MAY choose to cache the list of destination endpoints where this has occurred with and block use of EDO on future connections to those endpoints, but this cache MUST be accessible to users/applications on the host. Note that such endpoint assumptions can vary in the presence of load balancers where server implementations vary behind such balancers.

6.2. TCP States and Transitions

TCP EDO does not alter the existing TCP state or state transition mechanisms.

6.3. TCP Segment Processing

TCP EDO alters segment processing during the TCP option processing step. Once detected, the TCP EDO Extension option overrides the TCP Data Offset field for all subsequent option processing. Option processing continues at the next option (if present) after the EDO Extension option.

6.4. Impact on TCP Header Size

The TCP EDO Supported option increases SYN header length by a minimum of 2 bytes but could increase it by more depending on 32-bit word alignment. Currently popular SYN options total 19 bytes, which leaves more than enough room for the EDO Supported option:

- SACK permitted (2 bytes in SYN, optionally 2 + 8N bytes after) [RFC2018][RFC6675]

- Timestamp (10 bytes) [RFC7323]

- Window scale (3 bytes) [RFC7323]

- MSS option (4 bytes) [RFC9293]

Adding the EDO Supported option would result in a total of 21 bytes of SYN option space.

Subsequent segments would use 10 bytes of option space without any SACK blocks (TS only; WS and MSS are used only in SYN and SYN/ACK) or allow up to 3 SACK blocks before needing to use EDO; with EDO, the number of SACK blocks or additional options would be substantially increased. There are also other options that are emerging in the SYN, including TCP Fast Open, which uses another 6-18 (typically 10) bytes in the SYN/ACK of the first connection and in the SYN of subsequent connections [RFC7413].

TCP EDO can also be negotiated in SYNs with either of the following large options:

- TCP-AO (authentication) (16 bytes) [RFC5925]

- Multipath TCP (12 bytes in SYN and SYN/ACK, 20 after) [RFC8684]

Including TCP-AO with TS, WS, SACK increases the SYN option space use to 35 bytes; with Multipath TCP the use is 31 bytes. When Multipath TCP is enabled with the typical options, later segments would require 30 bytes without SACK, thus limiting the SACK option

to one block unless EDO is also supported on at least non-SYN segments.

The full combination of the above options (47 bytes for TS, WS, MSS, SACK, TCP-AO, and MPTCP) does not fit in the existing SYN option space and (as noted) that space cannot be extended within a single SYN segment. There has been a proposal to change TS to a 2 byte "TS permitted" signal in the initial SYN, provided it can be safely enabled during the connection later or might be avoided completely [Ni15]. Even using "TS-permitted", the total space is still too large to support in the initial SYN without SYN option space extension [Bo14][Br14][To22].

The EDO Extension option has negligible impact on other headers because it can either come first or just after security information, and in either case the additional 4 or 6 bytes are easily accommodated within the TCP Data Offset length. Once the EDO option is processed, the entirety of the remainder of the TCP segment is available for any remaining options.

6.5. Connectionless Resets

A RST may arrive during a currently active connection or may be needed to cleanup old state from an abandoned connection. The latter occurs when a new SYN is sent to an endpoint with matching existing connection state, at which point that endpoint responds with a RST and both ends remove stale information.

The EDO Extension option is mandatory on all TCP segments once negotiated, i.e., except in the SYN and SYN/ACK (which establish support) and the RST. A RST may lack the context to know that EDO is active on a connection.

>> The EDO Extension option MAY occur in a RST when the endpoint has connection state that has negotiated EDO. However, unless the RST is generated by an incoming segment that includes an EDO Extension option, the transmitted RST MUST NOT include the EDO Extension option.

6.6. ICMP Handling

ICMP responses are intended to include the IP and the port fields of TCP and UDP headers of typical TCP/IP and UDP/IP packets [RFC792]. This includes the first 8 data bytes of the original datagram, intended to include the transport port numbers used for connection demultiplexing. Later specifications encourage returning as much of the original payload as possible [RFC1812]. In either case, legacy

options or new options in the EDO extension area might or might not be included, and so options are generally not assumed to be part of ICMP processing anyway.

7. Interactions with Middleboxes

Middleboxes are on-path devices that typically examine or modify packets in ways that Internet routers do not [RFC3234]. This includes parsing transport headers and/or rewriting transport segments in ways that may affect EDO.

There are several cases to consider:

- Typical NAT/NAPT devices, which modify only IP address and/or TCP port number fields (with associated TCP checksum updates)

- Middleboxes that try to reconstitute TCP data streams, such as for deep-packet inspection for virus scanning

- Middleboxes that modify known TCP header fields

- Middleboxes that rewrite TCP segments

7.1. Middlebox Coexistence with EDO

Middleboxes can coexist with EDO when they either support EDO or when they ignore its impact on segment structure.

NATs and NAPTs, which rewrite IP address and/or transport port fields, are the most common form of middlebox and are not affected by the EDO option.

Middleboxes that support EDO would be those that correctly parse the EDO option. Such boxes can reconstitute the TCP data stream correctly or can modify header fields and/or rewrite segments without impact to EDO.

Conventional TCP proxies terminate the TCP connection in both directions and thus operate as TCP endpoints, such as when a client-middlebox and middlebox-server each have separate TCP connections. They would support EDO by following the host requirements herein on both connections. The use of EDO on one connection is independent of its use on the other in this case.

7.2. Middlebox Interference with EDO

Middleboxes that do not support EDO cannot coexist with its use when they modify segment boundaries or do not forward unknown (e.g., the EDO) options.

So-called "transparent" rewriting proxies, which inappropriately and incorrectly modify TCP segment boundaries, might mix option information with user data if they did not support EDO. Such devices might also interfere with other TCP options such as TCP-AO. There are three types of such boxes:

- Those that process received options and transmit sent options separately, i.e., although they rewrite segments, they behave as TCP endpoints in both directions.

- Those that split segments, taking a received segment and emitting two or more segments with revised headers.

- Those that join segments, receiving multiple segments and emitting a single segment whose data is the concatenation of the components.

In all three cases, EDO is either treated as independent on different sides of such boxes or not. If independent, EDO would either be correctly terminated in either or both directions or disabled due to lack of SYN/ACK confirmation in either or both directions. Problems would occur only when TCP segments with EDO are combined or split while ignoring the EDO option. In the split case, the key concern is if the split happens within the option extension space or if EDO is silently copied to both segments without copying the corresponding extended option space contents. However, the most comprehensive study of these cases indicates that "although middleboxes do split and coalesce segments, none did so while passing unknown options" [Ho11].

Note that the second and third types of middlebox behaviors listed above may create syndromes similar to TCP transmit and receive hardware offload engines that incorrectly modify segments with unknown options.

Middleboxes that silently remove options that they do not implement have been observed [Ho11]. Such boxes interfere with the use of the EDO Extension option in the SYN and SYN/ACK segments because extended option space would be misinterpreted as user data if the EDO Extension option were removed, and this cannot be avoided. This is one reason that SYN and SYN/ACK extension requires alternate

mechanisms (see Section 8.7). It is also the reason for the 6-byte EDO Extension variant (see Section 5.3), which can detect such merging or splitting of segments. Further, if such middleboxes become present on a path they could cause similar misinterpretation on segments exchanged in the ESTABLISHED and subsequent states. As a result, this document requires that the EDO Extension option be avoided on the SYN/ACK and that this option needs to be used on all segments once successfully negotiated and encourages use of the 6-byte EDO Extension variant.

Deep-packet inspection systems that inspect TCP segment payloads or attempt to reconstitute the data stream would incorrectly include option data in the reconstituted user data stream, which might interfere with their operation.

>> It can be important to detect misbehavior that could cause EDO space to be misinterpreted as user data. In such cases, EDO SHOULD be used in conjunction with an integrity protection mechanism. This includes the 6-byte EDO Extension variant or stronger mechanisms such as IPsec, TCP-AO, etc. It is useful to note that such protection only helps non-compliant components and enable avoidance (e.g., disabling EDO), but integrity protection alone cannot correct the misinterpretation of EDO space as user data.

This situation is similar to that of ECN and ICMP support in the Internet. In both cases, endpoints have evolved mechanisms for detecting and robustly operating around "black holes". Very similar algorithms are expected to be applicable for EDO.

8. Comparison to Previous Proposals

EDO is the latest in a long line of attempts to increase TCP option space [Al06][Ed08][Ko04][Ra12][Yo11]. The following is a comparison of these approaches to EDO, based partly on a previous summary [Ra12]. This comparison differs from that summary by using a different set of success criteria.

8.1. EDO Criteria

Our criteria for a successful solution are as follows:

- Zero-cost fallback to legacy endpoints.
- Minimal impact on middlebox compatibility.
- No additional side-effects.

Zero-cost fallback requires that upgraded hosts incur no penalty for attempting to use EDO. This disqualifies dual-stack approaches, because the client might have to delay connection establishment to wait for the preferred connection mode to complete. Note that the impact of legacy endpoints that silently reflect unknown options are not considered, as they are already non-compliant with existing TCP requirements [RFC9293].

Minimal impact on middlebox compatibility requires that EDO works through simple NAT and NAPT boxes, which modify IP addresses and ports and recompute IPv4 header and TCP segment checksums. Middleboxes that reject unknown options or that process segments in detail without regard for unknown options are not considered; they process segments as if they were an endpoint but do so in ways that are not compliant with existing TCP requirements (e.g., they should have rejected the initial SYN because of its unknown options rather than silently relaying it).

EDO also attempts to avoid creating side-effects, such as might happen if options were split across multiple TCP segments (which could arrive out of order or be lost) or across different TCP connections (which could fail to share fate through firewalls or NAT/NAPTs).

These requirements are similar to those noted in [Ra12], but EDO groups cases of segment modification beyond address and port - such as rewriting, segment drop, sequence number modification, and option stripping - as already in violation of existing TCP requirements regarding unknown options, and so we do not consider their impact on this new option.

8.2. Summary of Approaches

There are three basic ways in which TCP option space extension has been attempted:

1. Use of a TCP option.

2. Redefinition of the existing TCP header fields.

3. Use of option space in multiple TCP segments (split across multiple segments).

A TCP option is the most direct way to extend the option space and is the basis of EDO. This approach cannot extend the option space of the initial SYN.

Redefining existing TCP header fields can be used to either contain additional options or as a pointer indicating alternate ways to interpret the segment payload. All such redefinitions make it difficult to achieve zero-impact backward compatibility, both with legacy endpoints and middleboxes.

Splitting option space across separate segments can create unintended side-effects, such as increased delay to deal with path latency or loss differences.

The following discusses three of the most notable past attempts to extend the TCP option space: Extended Segments, TCPx2, LO/SLO, and LOIC. [Ra12] suggests a few other approaches, including use of TCP option cookies, reuse/overload of other TCP fields (e.g., the URG pointer), or compressing TCP options. None of these is compatible with legacy endpoints or middleboxes.

8.3. Extended Segments

TCP Extended Segments redefined the meaning of currently unused values of the Data Offset (DO) field [Ko04]. TCP defines DO as indicating the length of the TCP header, including options, in 32-bit words. The default TCP header with no options is 5 such words, so the minimum currently valid DO value is 5 (meaning 40 bytes of option space). This document defines interpretations of values 0-4: DO=0 means 48 bytes of option space, DO=1 means 64, DO=2 means 128, DO=3 means 256, and DO=4 means unlimited (e.g., the entire payload is option space). This variant negotiates the use of this capability by using one of these invalid DO values in the initial SYN.

Use of this variant is not backward-compatible with legacy TCP implementations, whether at the desired endpoint or on middleboxes. The variant also defines a way to initiate the feature on the passive side, e.g., using an invalid DO during the SYN/ACK when the initial SYN had a valid DO. This capability allows either side to initiate use of the feature but is also not backward compatible.

8.4. TCPx2

TCPx2 redefines legacy TCP headers by basically doubling all TCP header fields [Al06]. It relies on a new transport protocol number to indicate its use, defeating backward compatibility with all existing TCP capabilities, including firewalls, NATs/NAPTs, and legacy endpoints and applications.

8.5. LO/SLO

The TCP Long Option (LO, [Ed08]) is very similar to EDO, except that presence of LO results in ignoring the existing Data Offset (DO) field and that LO is required to be the first option. EDO considers the need for other fields to be first and declares that the EDO is the last option as indicated by the DO field value. Like LO, EDO is required in every segment once negotiated.

The TCP Long Option draft also specified the SYN Long Option (SLO) [Ed08]. If SLO is used in the initial SYN and successfully negotiated, it is used in each subsequent segment until all of the initial SYN options are transmitted.

LO is backward compatible, as is SLO; in both cases, endpoints not supporting the option would not respond with the option, and in both cases the initial SYN is not itself extended.

SLO does modify the three-way handshake because the connection isn't considered completely established until the first data byte is acknowledged. Legacy TCP can establish a connection even in the absence of data. SLO also changes the semantics of the SYN/ACK; for legacy TCP, this completes the active side connection establishment, where in SLO an additional data ACK is required. A connection whose initial SYN options have been confirmed in the SYN/ACK might still fail upon receipt of additional options sent in later SLO segments. This case - of late negotiation fail - is not addressed in the specification.

8.6. LOIC

TCP Long Options by Invalid Checksum is a dual-stack approach that uses two initial SYNS to initiate all updated connections [Yo11]. One SYN negotiates the new option and the other SYN payload contains only the entire options. The negotiation SYN is compliant with existing procedures, but the option SYN has a deliberately incorrect TCP checksum (decremented by 2). A legacy endpoint would discard the segment with the incorrect checksum and respond to the negotiation SYN without the LO option.

Use of the option SYN and its incorrect checksum both interfere with other legacy components. Segments with incorrect checksums will be silently dropped by most middleboxes, including NATs/NAPTs. Use of two SYNs creates side-effects that can delay connections to upgraded endpoints, notably when the option SYN is lost or the SYNs arrive out of order. Finally, by not allowing other options in the negotiation SYN, all connections to legacy endpoints either use no

options or require a separate connection attempt (either concurrent or subsequent).

8.7. Problems with Extending the Initial SYN

The key difficulty with most previous proposals is the desire to extend the option space in all TCP segments, including the initial SYN, i.e., SYN with no ACK, typically the first segment of a connection, as well as possibly the SYN/ACK. It has proven difficult to extend space within the segment of the initial SYN in the absence of prior negotiation while maintaining current TCP three-way handshake properties, and it may be similarly challenging to extend the SYN/ACK (depending on asymmetric middlebox assumptions).

A new TCP option cannot extend the Data Offset of a single TCP initial SYN segment and cannot extend a SYN/ACK in a single segment when considering misbehaving middleboxes. All TCP segments, including the initial SYN and SYN/ACK, may include user data in the payload data [RFC9293], and this can be useful for some proposed features such as TCP Fast Open [RFC7413]. Legacy endpoints that ignore the new option would process the payload contents as user data and send an ACK. Once ACK'd, this data cannot be removed from the user stream.

The Reserved TCP header bits cannot be redefined easily, even though three of the six total bits have already been redefined (ECE/CWR [RFC3168] and NS [RFC3540]). Legacy endpoints have been known to reflect received values in these fields; this was safely dealt with for ECN but would be difficult here [RFC3168].

TCP initial SYN (SYN and not ACK) segments can use every other TCP header field except the Acknowledgement number, which is not used because the ACK field is not set. In all other segments, all fields except the three remaining Reserved header bits are actively used. The total amount of available header fields, in either case, is insufficient to be useful in extending the option space.

The representation of TCP options can be optimized to minimize the space needed. In such cases, multiple Kind and Length fields are combined, so that a new Kind would indicate a specific combination of options, whose order is fixed and whose length is indicated by one Length field. Most TCP options use fields whose size is much larger than the required Kind and Length components, so the resulting efficiency is typically insufficient for additional options.

The option space of an initial SYN segment might be extended by using multiple initial segments (e.g., multiple SYNs or a SYN and non-SYN) or based on the context of previous or parallel connections. This method may also be needed to extend space in the SYN/ACK in the presence of misbehaving middleboxes. Because of their potential complexity, these approaches are addressed in separate documents [Bo14][Br14][To22].

Option space cannot be extended in outer layer headers, e.g., IPv4 or IPv6. These layers typically try to avoid extensions altogether, to simplify forwarding processing at routers. Introducing new shim layers to accommodate additional option space would interfere with deep-packet inspection mechanisms that are in widespread use.

As a result, EDO does not attempt to extend the space available for options in TCP initial SYNs. It does extend that space in all other segments (including SYN/ACK), which has always been trivially possible once an option is defined.

9. Implementation Issues

TCP segment processing can involve accessing nonlinear data structures, such as chains of buffers. Such chains are often designed so that the maximum default TCP header (60 bytes) fits in the first buffer. Extending the TCP header across multiple buffers may necessitate buffer traversal functions that span boundaries between buffers. Such traversal can also have a significant performance impact, which is additional rationale for using TCP option space - even extended option space - sparingly.

Although EDO can be large enough to consume the entire segment, it is important to leave space for data so that the TCP connection can make forward progress. It would be wise to limit EDO to consuming no more than MSS-4 bytes of the IP segment, preferably even less (e.g., MSS-128 bytes).

When using the ExID variant for testing and experimentation, either TCP option codepoint (253, 254) is valid in sent or received segments.

Some TCP segment offload devices (TSOs) incorrectly attempt to process segments with unknown options. EDO data needs to be passed to the protocol stack as part of the option space, not integrated with the user segment, to allow the offload to independently determine user data segment boundaries and combine them correctly with the extended option data. Issues with incorrect resegmentation

by an offload engine can be detected in the same way as middlebox tampering.

10. Security Considerations

It is meaningless to have the Data Offset further exceed the position of the EDO data offset option.

>> When the EDO Extension option is present, the EDO Extension option SHOULD be the last non-null option covered by the TCP Data Offset, because it would be the last option affected by Data Offset.

This also makes it more difficult to use the Data Offset field as a covert channel.

11. IANA Considerations

We request that, upon publication, this option be assigned a TCP Option codepoint by IANA, which the RFC Editor will replace EDO-OPT in this document with codepoint value.

The TCP Experimental ID (ExID) with a 16-bit value of 0x0ED0 (in network standard byte order) has been assigned for use during testing and preliminary experiments.

12. References

12.1. Normative References

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

[RFC9293] Eddy, W. (Ed.), "Transmission Control Protocol", STD 7, RFC 9293, Aug. 2022.

12.2. Informative References

[Al06] Allman, M., "TCPx2: Don’t Fence Me In", draft-allman-tcpx2-hack-00 (work in progress), May 2006.

[Bo14] Borman, D., "TCP Four-Way Handshake", draft-borman-tcp4way-00 (work in progress), October 2014.

[Br14] Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[Ho11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP", Proc. ACM Sigcomm Internet Measurement Conference (IMC), 2011, pp. 181-194.

[Ko04] Kohler, E., "Extended Option Space for TCP", draft-kohler-tcpm-extopt-00 (work in progress), September 2004.

[Ni15] Nishida, Y., "A-PAWS: Alternative Approach for PAWS", draft-nishida-tcpm-apaws-02 (work in progress), Oct. 2015.

[Ra12] Ramaiah, A., "TCP option space extension", draft-ananth-tcpm-tcpoptext-00 (work in progress), March 2012.

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792, September 1981.

[RFC1812] Baker, F. (Ed.), "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

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

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

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

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

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, Apr. 2020.

[To22] Touch, J., T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt (work in progress), Oct. 2022.

[Yo11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

13. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback, in particular: Oliver Bonaventure, Bob Briscoe, Ted Faber, John Leslie, Pasi Sarolahti, Michael Scharf, Richard Scheffenegger, and Alexander Zimmerman.

This work was partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors’ Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Wesley M. Eddy
MTI Systems
US

Email: wes@mti-systems.com

TCPM
Internet-Draft
Intended status: Standards Track
Expires: 28 April 2022

M. Scharf
Hochschule Esslingen
M. Jethanandani
Kloud Services
V. Murgai
Samsung

25 October 2021

YANG Model for Transmission Control Protocol (TCP) Configuration
draft-ietf-tcpm-yang-tcp-04

Abstract

This document specifies a minimal YANG model for TCP on devices that are configured by network management protocols. The YANG model defines a container for all TCP connections and groupings of authentication parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model also includes basic TCP statistics. The model is NMDA (RFC 8342) compliant.

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 https://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 28 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Requirements Language 4
  2.1. Note to RFC Editor 4
3. Model Overview 4
  3.1. Modeling Scope 4
  3.2. Model Design 6
  3.3. Tree Diagram 6
4. TCP YANG Model 6
5. IANA Considerations 13
  5.1. The IETF XML Registry 13
  5.2. The YANG Module Names Registry 14
6. Security Considerations 14
7. References 15
  7.1. Normative References 15
  7.2. Informative References 17
Appendix A. Acknowledgements 18
Appendix B. Changes compared to previous versions 18
Appendix C. Examples 19
  C.1. Keepalive Configuration 19
  C.2. TCP-AO Configuration 20
Appendix D. Complete Tree Diagram 21
Authors' Addresses 22

1. Introduction

The Transmission Control Protocol (TCP) Specification [I-D.ietf-tcpm-rfc793bis] is used by many applications in the Internet, including control and management protocols. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a minimal YANG [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. The model has a narrow scope and focuses on a subset of fundamental TCP functions and basic statistics. It defines a container for TCP connection that includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model also enables configuration of TCP-AO [RFC5925], which is a relevant TCP feature on network elements such as routers. The model can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. Moreover, many existing TCP/IP stacks do not use YANG data models. Such TCP implementations often have other means to configure the parameters listed in this document, which are outside the scope of this document.

This specification is orthogonal to the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. It is possible also to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are orthogonal to this specification. Examples are:

* TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [RFC8783].

* TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2021-10-25 with the actual date of the publication of this document.

3. Model Overview

3.1. Modeling Scope

TCP is implemented on many different system architectures. As a result, there are may different and often implementation-specific ways to configure parameters of the TCP protocol engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

* Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.

* Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.

* Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are

socket options in the socket API, such as disabling the Nagle algorithm by TCP_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

* Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementation have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks.

This in particular applies to the support of TCP-AO [RFC5925]. TCP Authentication Option (TCP-AO) is used on routers to secure routing protocols such as BGP. In that case, a YANG model for TCP-AO configuration is required. The model defined in this document includes the required parameters for TCP-AO configuration, such as the values of SendID and RecvID. The key chain for TCP-AO can be modeled by the YANG Data Model for Key Chains [RFC8177].

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. As the TCP MD5 signature option is obsoleted by TCP-AO, it is strongly RECOMMENDED to use TCP-AO.

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics and a TCP connection table.

* Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information

* TCP connection table: Access to status information for all TCP connections. Note, the connection table is modeled as a list that is read-writeable, even though a connection cannot be created by adding entries to the table. Similarly, deletion of connections from this list is implementation-specific.

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo. Note that the TCP MIB does not include means to reset statistics, which are defined in this document. This is not a major addition, as a reset can simply be implemented by storing offset values for the counters.

3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
module: ietf-tcp
  +--rw tcp!
     +--rw connections
     |     ...
     +--ro statistics {statistics}?
           ...
```

4. TCP YANG Model

```
<CODE BEGINS> file "ietf-tcp@2021-10-25.yang"
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-yang-types {
    prefix "yang";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  organization
    "IETF TCPM Working Group";

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/tcpm/about>
     WG List:  <tcpm@ietf.org>

     Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
              Mahesh Jethanandani (mjethanandani at gmail dot com)
              Vishal Murgai (vmurgai at gmail dot com)";

  description
    "This module focuses on fundamental TCP functions and basic
     statistics. The model can be augmented to address more advanced
     or implementation specific TCP features.

     Copyright (c) 2021 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC XXXX
     (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
     for full legal notices.
```

```
        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

     revision "2021-10-25" {
       description
         "Initial Version";
       reference
         "RFC XXXX, YANG Model for Transmission Control Protocol (TCP)
                    Configuration.";
     }

     // Features
     feature statistics {
       description
         "This implementation supports statistics reporting.";
     }

     // TCP-AO Groupings

     grouping ao {
       leaf enable-ao {
         type boolean;
         default "false";
         description
           "Enable support of TCP-Authentication Option (TCP-AO).";
       }

       leaf send-id {
         type uint8 {
           range "0..255";
         }
         must "../enable-ao = 'true'";
         description
           "The SendID is inserted as the KeyID of the TCP-AO option
            of outgoing segments. The SendID must match the RecvID
            at the other endpoint.";
         reference
           "RFC 5925: The TCP Authentication Option.";
       }

       leaf recv-id {
         type uint8 {
           range "0..255";
         }
         must "../enable-ao = 'true'";
```

```
        description
          "The RecvID is matched against the TCP-AO KeyID of incoming
           segments. The RecvID must match the SendID at the other
           endpoint.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      leaf include-tcp-options {
        type boolean;
        must "../enable-ao = 'true'";
        default true;
        description
          "Include TCP options in MAC calculation.";
      }

      leaf accept-key-mismatch {
        type boolean;
        must "../enable-ao = 'true'";
        description
          "Accept TCP segments with a Master Key Tuple (MKT) that is
           not configured.";
      }
      description
        "Authentication Option (AO) for TCP.";
      reference
        "RFC 5925: The TCP Authentication Option.";
    }

    // MD5 grouping

    grouping md5 {
      description
        "Grouping for use in authenticating TCP sessions using MD5.";
      reference
        "RFC 2385: Protection of BGP Sessions via the TCP MD5
         Signature.";

      leaf enable-md5 {
        type boolean;
        default "false";
        description
          "Enable support of MD5 to authenticate a TCP session.";
      }
    }

    // TCP configuration
```

```
     container tcp {
       presence "The container for TCP configuration.";

       description
         "TCP container.";

       container connections {
         list connection {
           key "local-address remote-address local-port remote-port";

           leaf local-address {
             type inet:ip-address;
             description
               "Local address that forms the connection identifier.";
           }

           leaf remote-address {
             type inet:ip-address;
             description
               "Remote address that forms the connection identifier.";
           }

           leaf local-port {
             type inet:port-number;
             description
               "Local TCP port that forms the connection identifier.";
           }

           leaf remote-port {
             type inet:port-number;
             description
               "Remote TCP port that forms the connection identifier.";
           }

           container common {
             uses tcpcmn:tcp-common-grouping;

             choice authentication {
               case ao {
                 uses ao;
                 description
                   "Use TCP-AO to secure the connection.";
               }

               case md5 {
                 uses md5;
                 description
                   "Use TCP-MD5 to secure the connection.";
```

```
              }
              description
                "Choice of how to secure the TCP connection.";
            }
            description
              "Common definitions of TCP configuration. This includes
               parameters such as how to secure the connection,
               that can be part of either the client or server.";
          }
          description
            "List of TCP connections with their parameters. The list
             is modeled as writeable, but implementations may not
             allow creation of new TCP connections by adding entries to
             the list. Furthermore, the behavior upon removal is
             implementation-specific. Implementations may support
             closing or resetting a TCP connection upon an operation
             that removes the entry from the list.";
        }
        description
          "A container of all TCP connections.";
      }

      container statistics {
        if-feature statistics;
        config false;

        leaf active-opens {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a
             direct transition to the SYN-SENT state from the CLOSED
             state.";
        }

        leaf passive-opens {
          type yang:counter32;
          description
            "The number of times TCP connections have made a direct
             transition to the SYN-RCVD state from the LISTEN state.";
        }

        leaf attempt-fails {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a
             direct transition to the CLOSED state from either the
             SYN-SENT state or the SYN-RCVD state, plus the number of
             times that TCP connections have made a direct transition
```

```
             to the LISTEN state from the SYN-RCVD state.";
         }

         leaf establish-resets {
           type yang:counter32;
           description
             "The number of times that TCP connections have made a
              direct transition to the CLOSED state from either the
              ESTABLISHED state or the CLOSE-WAIT state.";
         }

         leaf currently-established {
           type yang:gauge32;
           description
             "The number of TCP connections for which the current state
              is either ESTABLISHED or CLOSE-WAIT.";
         }

         leaf in-segments {
           type yang:counter64;
           description
             "The total number of segments received, including those
              received in error.  This count includes segments received
              on currently established connections.";
         }

         leaf out-segments {
           type yang:counter64;
           description
             "The total number of segments sent, including those on
              current connections but excluding those containing only
              retransmitted octets.";
         }

         leaf retransmitted-segments {
           type yang:counter32;
           description
             "The total number of segments retransmitted; that is, the
              number of TCP segments transmitted containing one or more
              previously transmitted octets.";
         }

         leaf in-errors {
           type yang:counter32;
           description
             "The total number of segments received in error (e.g., bad
              TCP checksums).";
         }
```

```
        leaf out-resets {
          type yang:counter32;
          description
            "The number of TCP segments sent containing the RST flag.";
        }

        action reset {
          description
            "Reset statistics action command.";
          input {
            leaf reset-at {
              type yang:date-and-time;
              description
                "Time when the reset action needs to be
                 executed.";
            }
          }
          output {
            leaf reset-finished-at {
              type yang:date-and-time;
              description
                "Time when the reset action command completed.";
            }
          }
        }
        description
          "Statistics across all connections.";
      }
    }
  }
  <CODE ENDS>
```

5. IANA Considerations

5.1. The IETF XML Registry

This document registers an URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

This document registers a YANG module in the "YANG Module Names" registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

```
name:         ietf-tcp
namespace:    urn:ietf:params:xml:ns:yang:ietf-tcp
prefix:       tcp
reference:    RFC XXXX
```

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

* Common configuration included from NETCONF Client and Server Models [I-D.ietf-netconf-tcp-client-server]. Unrestricted access to all the nodes, e.g., keepalive idle-timer, can cause connections to fail or to timeout prematurely.

* Authentication configuration. Unrestricted access to the nodes under authentication configuration can prevent the use of authenticated communication and cause connection setups to fail. This can result in massive security vulnerabilities and service disruption for the traffic requiring authentication.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

* Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack, e.g. MITM.

* Similarly, unrestricted access to statistics of the client or server can be used by a malicious user to exploit any vulnerabilities of the system.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

* The YANG module allows for the statistics to be cleared by executing the reset action. This action should be restricted to users with the right permission.

7. References

7.1. Normative References

[I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-10, 18 May 2021, <https://www.ietf.org/archive/id/draft-ietf-netconf-tcp-client-server-10.txt>.

[I-D.ietf-tcpm-rfc793bis]
Eddy, W. M., "Transmission Control Protocol (TCP) Specification", Work in Progress, Internet-Draft, draft-ietf-tcpm-rfc793bis-25, 7 September 2021, <https://www.ietf.org/archive/id/draft-ietf-tcpm-rfc793bis-25.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <https://www.rfc-editor.org/info/rfc8177>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

7.2. Informative References

[I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-model-11, 11 July 2021, <https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-11.txt>.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P. S., Wood, C. A., Pauly, T., and K. Rose, "An Abstract Application Layer Interface to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-interface-13, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-taps-interface-13.txt>.

[I-D.touch-tcpm-ao-test-vectors]
Touch, J. and J. Kuusisaari, "TCP-AO Test Vectors", Work in Progress, Internet-Draft, draft-touch-tcpm-ao-test-vectors-02, 23 December 2020, <https://www.ietf.org/archive/id/draft-touch-tcpm-ao-test-vectors-02.txt>.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <https://www.rfc-editor.org/info/rfc4022>.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <https://www.rfc-editor.org/info/rfc4898>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <https://www.rfc-editor.org/info/rfc6643>.

[RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <https://www.rfc-editor.org/info/rfc8512>.

[RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <https://www.rfc-editor.org/info/rfc8513>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

[RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <https://www.rfc-editor.org/info/rfc8783>.

Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews: Mohamed Boucadair

Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-04

* Removed congestion control

* Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

* Updated TCP-AO grouping

* Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

* Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list

* Editorial bugfixes and outdated references reported by Mohamed Boucadair

* Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

* Alignment with [I-D.ietf-netconf-tcp-client-server]

* Removing backward-compatibility to the TCP MIB

* Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

* Editorial improvements

Appendix C. Examples

C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

[note: '\' line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example shows how TCP keepalive can be configured for
a given connection. An idle connection is dropped after
idle-time + (max-probes * probe-interval).
-->
<tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
  <connections>
    <connection>
      <local-address>192.168.1.1</local-address>
      <remote-address>192.168.1.2</remote-address>
      <local-port>1025</local-port>
      <remote-port>80</remote-port>
      <common>
        <keepalives>
          <idle-time>5</idle-time>
          <max-probes>5</max-probes>
          <probe-interval>10</probe-interval>
        </keepalives>
      </common>
    </connection>
  </connections>
</tcp>
```

C.2. TCP-AO Configuration

The following example demonstrates how to model a TCP-AO [RFC5925] configuration for the example in TCP-AO Test Vectors [I-D.touch-tcpm-ao-test-vectors], Section 5.1.1.

[note: '\' line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example sets TCP-AO configuration parameters as
demonstrated by examples in draft-touch-tcpm-ao-test-vectors.
-->

<tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
  <connections>
    <connection>
      <local-address>192.168.1.1</local-address>
      <remote-address>192.168.1.2</remote-address>
      <local-port>1025</local-port>
      <remote-port>80</remote-port>
      <common>
        <enable-ao>true</enable-ao>
      </common>
    </connection>
  </connections>
</tcp>

<key-chains
    xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
  <key-chain>
    <name>ao-config</name>
    <description>"An example for TCP-AO configuration."</description>\

    <key>
      <key-id>61</key-id>
      <crypto-algorithm>hmac-sha-1</crypto-algorithm>
      <key-string>
        <hexadecimal-string>01:23:a5:93:b9:db:70:62:9b:be:2c:a6:77:cd:fd:ea:\
6f:e0:ac:ad</hexadecimal-string>
      </key-string>
    </key>
  </key-chain>
</key-chains>
```

Appendix D. Complete Tree Diagram

Here is the complete tree diagram for the TCP YANG model.

```
module: ietf-tcp
  +--rw tcp!
     +--rw connections
     |  +--rw connection*
     |          [local-address remote-address local-port remote-port]
     |     +--rw local-address     inet:ip-address
     |     +--rw remote-address    inet:ip-address
     |     +--rw local-port        inet:port-number
     |     +--rw remote-port       inet:port-number
     |     +--rw common
     |        +--rw keepalives!
     |        |  +--rw idle-time         uint16
     |        |  +--rw max-probes        uint16
     |        |  +--rw probe-interval    uint16
     |        +--rw (authentication)?
     |           +--:(ao)
     |           |  +--rw enable-ao?             boolean
     |           |  +--rw send-id?               uint8
     |           |  +--rw recv-id?               uint8
     |           |  +--rw include-tcp-options?   boolean
     |           |  +--rw accept-key-mismatch?   boolean
     |           +--:(md5)
     |              +--rw enable-md5?            boolean
     +--ro statistics {statistics}?
        +--ro active-opens?             yang:counter32
        +--ro passive-opens?            yang:counter32
        +--ro attempt-fails?            yang:counter32
        +--ro establish-resets?         yang:counter32
        +--ro currently-established?    yang:gauge32
        +--ro in-segments?              yang:counter64
        +--ro out-segments?             yang:counter64
        +--ro retransmitted-segments?   yang:counter32
        +--ro in-errors?                yang:counter32
        +--ro out-resets?               yang:counter32
        +---x reset
           +---w input
           |  +---w reset-at?   yang:date-and-time
           +--ro output
              +--ro reset-finished-at?   yang:date-and-time
```

Authors' Addresses

Michael Scharf
Hochschule Esslingen - University of Applied Sciences
Flandernstr. 101
73732 Esslingen
Germany

Email: michael.scharf@hs-esslingen.de

Mahesh Jethanandani
Kloud Services

Email: mjethanandani@gmail.com

Vishal Murgai
Samsung

Email: vmurgai@gmail.com

TCPM
Internet-Draft
Intended status: Standards Track
Expires: 15 March 2023

M. Scharf
Hochschule Esslingen
M. Jethanandani
Kloud Services
V. Murgai
Samsung

11 September 2022

A YANG Model for Transmission Control Protocol (TCP) Configuration and State

draft-ietf-tcpm-yang-tcp-09

Abstract

This document specifies a minimal YANG model for TCP on devices that are configured and managed by network management protocols. The YANG model defines a container for all TCP connections, and groupings of authentication parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model also includes basic TCP statistics. The model is compliant with Network Management Datastore Architecture (NMDA) (RFC 8342).

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 https://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 15 March 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 2
2. Requirements Language 4
  2.1. Note to RFC Editor 4
3. YANG Module Overview 4
  3.1. Scope 4
  3.2. Model Design 6
  3.3. Tree Diagram 6
4. TCP YANG Model 7
5. IANA Considerations 19
  5.1. The IETF XML Registry 19
  5.2. The YANG Module Names Registry 20
6. Security Considerations 20
7. References 21
  7.1. Normative References 21
  7.2. Informative References 23
Appendix A. Acknowledgements 25
Appendix B. Examples 25
  B.1. Keepalive Configuration 26
  B.2. TCP-AO Configuration 26
Appendix C. Complete Tree Diagram 28
Authors' Addresses 29

1. Introduction

The Transmission Control Protocol (TCP) [RFC9293] is used by many applications in the Internet, including control and management protocols. As such, TCP is implemented on network elements that can be configured and managed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].

This document specifies a minimal YANG 1.1 [RFC7950] model for configuring and managing TCP on network elements that support YANG, a TCP connection table, a TCP listener table containing information about a particular TCP listener, and an augmentation of the YANG Data Model for Key Chains [RFC8177] to support authentication. The YANG module specified in this document is compliant with Network Management Datastore Architecture (NMDA) [RFC8342].

The YANG module has a narrow scope and focuses on a subset of fundamental TCP functions and basic statistics. It defines a container for a list of TCP connections that includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model adheres to the recommendation in BGP/MPLS IP Virtual Private Networks [RFC4364]. Therefore it allows enabling of TCP-AO [RFC5925], and accommodates the installed base that makes use of MD5. The module can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

This specification does not deprecate the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. The YANG module also omits some selected parameters included in TCP MIB, most notably Retransmission Timeout (RTO) configuration and a maximum connection limit. This is a conscious decision as these parameters hardly matter in a state-of-the-art TCP implementation. It would also be possible to translate a MIB into a YANG module, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, because a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are orthogonal to this specification. Examples are:

* TCP header attributes are modeled in other security-related models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519], Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [RFC8783], I2NSF Capability YANG Data Model [I-D.ietf-i2nsf-capability-data-model], or I2NSF Network Security Function-Facing Interface YANG Data Model [I-D.ietf-i2nsf-nsf-facing-interface-dm].

* TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

* TCP-AO and TCP MD5 configuration for Layer 3 VPNs is modeled in A Layer 3 VPN Network YANG Model [RFC9182]. This model assumes that TCP-AO specific parameters are preconfigured in addition to the keychain parameters.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2022-09-11 with the actual date of the publication of this document.

3. YANG Module Overview

3.1. Scope

TCP is implemented on different system architectures. As a result, there are many different and often implementation-specific ways to configure parameters of the TCP engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

* System-wide configuration: Many TCP implementations have configuration parameters that affect all TCP connections from or to this TCP stack. Typical examples include enabling or disabling optional protocol features. For instance, many implementations can turn on or off use of window scaling Transmission Control Protocol (TCP) Specification [RFC9293] for all TCP connections.

* Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.

* Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. Typical examples are socket options in the socket API, such as disabling the Nagle algorithm Transmission Control Protocol (TCP) Specification [RFC9293] by TCP_NODELAY. If an application uses such an interface, it is possible that the configuration of the

application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

* Application preferences: Setting of TCP parameters can also be part of application preferences, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementations have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters. An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks.

This in particular applies to the support of TCP-AO [RFC5925] and the corresponding cryptographic algorithms [RFC5926]. TCP Authentication Option (TCP-AO) is used on routers to secure routing protocols such as BGP. In that case, a YANG model for TCP-AO configuration is required. The model defined in this document includes the required parameters for TCP-AO configuration, such as the values of SendID and RecvID. The keychain for TCP-AO can be modeled by the YANG Data Model for Key Chains [RFC8177]. The groupings defined in this document can be imported and used as part of such a preconfiguration.

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. The TCP MD5 signature option was obsoleted by TCP-AO in 2010. If current implementations require TCP authentication, it is RECOMMENDED to use TCP-AO [RFC5925].

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics, a TCP connection list, and a TCP listener list.

* Statistics: Counters for the number of active/passive opens, sent and received TCP segments, errors, and possibly other detailed debugging information

* TCP connection list: Access to status information for all TCP connections. Note, the connection table is modeled as a list that is read-writeable, even though a connection cannot be created by adding entries to the table. Similarly, deletion of connections from this list is implementation-specific.

* TCP listener list: A list containing information about TCP listeners, i.e., applications willing to accept connections.

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo. Note that the TCP MIB does not include means to reset statistics, which are defined in this document. This is not a major addition, as a reset can simply be implemented by storing offset values for the counters.

This version of the module does not model details of Multipath TCP [RFC8684]. This could be addressed in a later version of this document.

3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

3.3. Tree Diagram

This section provides an abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340]. A complete tree diagram can be found in the Appendix.

```
module: ietf-tcp
  +--rw tcp!
     +--rw connections
     |     ...
     +--ro tcp-listeners* [type address port]
     |     ...
     +--ro statistics {statistics}?
           ...

  augment /key-chain:key-chains/key-chain:key-chain/key-chain:key:
    +--rw authentication {authentication}?
       +--rw keychain?    key-chain:key-chain-ref
       +--rw (authentication)?
             ...
```

4. TCP YANG Model

This YANG module references The TCP Authentication Option [RFC5925], Protection of BGP Sessions via the TCP MD5 Signature [RFC2385], Transmission Control Protocol (TCP) Specification [RFC9293], and imports Common YANG Data Types [RFC6991], The NETCONF Access Control Model [RFC8341], and YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server].

```
<CODE BEGINS> file "ietf-tcp@2022-09-11.yang"
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-yang-types {
    prefix "yang";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
    reference
      "I-D.ietf-netconf-tcp-client-server: YANG Groupings for TCP
       Clients and TCP Servers.";
  }
  import ietf-inet-types {
    prefix "inet";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-netconf-acm {
    prefix nacm;
```

```
      reference
        "RFC 8341: Network Configuration Access Control Model";
    }
    import ietf-key-chain {
      prefix key-chain;
      reference
        "RFC 8177: YANG Key Chain.";
    }

    organization
      "IETF TCPM Working Group";

    contact
      "WG Web:   <https://datatracker.ietf.org/wg/tcpm/about>
       WG List:  <tcpm@ietf.org>

       Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
                Mahesh Jethanandani (mjethanandani at gmail dot com)
                Vishal Murgai (vmurgai at gmail dot com)";

    description
      "This module focuses on fundamental TCP functions and basic
       statistics. The model can be augmented to address more advanced
       or implementation specific TCP features.

       Copyright (c) 2022 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Simplified BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in this document are to be interpreted as
       described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
       they appear in all capitals, as shown here.";

    revision "2022-09-11" {
      description
        "Initial Version";
```

```
      reference
        "RFC XXXX, A YANG Model for Transmission Control Protocol (TCP)
                   Configuration and State.";
    }

    // Typedefs
    typedef mss {
      type uint16;
      description
        "Type definition for Maximum Segment Size.";
    }

    // Features
    feature statistics {
      description
        "This implementation supports statistics reporting.";
    }

    feature authentication {
      description
        "This implementation supports authentication.";
    }

    // Identities
    identity aes-128 {
      base key-chain:crypto-algorithm;
      description
        "AES128 authentication algorithm used by TCP-AO.";
      reference
         "RFC 5926: Cryptographic Algorithms for the TCP
          Authentication Option (TCP-AO).";
    }

    // TCP-AO Groupings

    grouping ao {
      leaf send-id {
        type uint8 {
          range "0..max";
        }
        description
          "The SendID is inserted as the KeyID of the TCP-AO option
           of outgoing segments. In a consistent configuration, the
           SendID matches the RecvID at the other endpoint.";
        reference
          "RFC 5925: The TCP Authentication Option, Section 3.1.";
      }
```

```
      leaf recv-id {
        type uint8 {
          range "0..max";
        }
        description
          "The RecvID is matched against the TCP-AO KeyID of incoming
           segments. In a consistent configuration, the RecvID matches
           the SendID at the other endpoint.";
        reference
          "RFC 5925: The TCP Authentication Option, Section 3.1.";
      }

      leaf include-tcp-options {
        type boolean;
        default true;
        description
          "When set to true, TCP options are included in MAC
           calculation.";
        reference
          "RFC 5925: The TCP Authentication Option, Section 3.1.";
      }

      leaf accept-key-mismatch {
        type boolean;
        description
          "Accept, when set to true, TCP segments with a Master Key
           Tuple (MKT) that is not configured.";
        reference
          "RFC 5925: The TCP Authentication Option, Section 7.3.";
      }

      leaf r-next-key-id {
        type uint8;
        config false;
        description
          "A field indicating the Master Key Tuple (MKT) that is ready
           at the sender to be used to authenticate received segments,
           i.e., the desired 'receive next' key ID.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      description
        "Authentication Option (AO) for TCP.";
      reference
        "RFC 5925: The TCP Authentication Option.";
    }
```

```
     // TCP configuration

     container tcp {
       presence "The container for TCP configuration.";

       description
         "TCP container.";

       container connections {
         list connection {
           key "local-address remote-address local-port remote-port";

           leaf local-address {
             type inet:ip-address;
             description
               "Identifies the address that is used by the local
                endpoint for the connection, and is one of the four
                elements that form the connection identifier.";
           }

           leaf remote-address {
             type inet:ip-address;
             description
               "Identifies the address that is used by the remote
                endpoint for the connection, and is one of the four
                elements that form the connection identifier.";
           }

           leaf local-port {
             type inet:port-number;
             description
               "Identifies the local TCP port used for the connection,
                and is one of the four elements that form the
                connection identifier.";
           }

           leaf remote-port {
             type inet:port-number;
             description
               "Identifies the remote TCP port used for the connection,
                and is one of the four elements that form the
                connection identifier.";
           }

           leaf mss {
             type mss;
             description
               "Maximum Segment Size (MSS) desired on this connection.
```

```
                Note, the 'effective send MSS' can be smaller than
                what is configured here.";
            reference
              "RFC 9293: Transmission Control Protocol (TCP)
                         Specification.";
          }

          leaf pmtud {
            type boolean;
            default false;
            description
              "Turns Path Maximum Transmission Unit Discovery (PMTUD)
               on (true) or off (false).";
            reference
              "RFC 9293: Transmission Control Protocol (TCP)
                         Specification.";
          }

          uses tcpcmn:tcp-common-grouping;

          leaf state {
            type enumeration {
              enum closed {
                value 1;
                description
                  "Connection is closed. Connections in this state
                   may not appear in this list.";
              }
              enum listen {
                value 2;
                description
                  "Represents waiting for a connection request from any
                   remote TCP peer and port.";
              }
              enum syn-sent {
                value 3;
                description
                  "Represents waiting for a matching connection request
                   after having sent a connection request.";
              }
              enum syn-received {
                value 4;
                description
                  "Represents waiting for a confirming connection
                   request acknowledgment after having both received
                   and sent a connection request.";
              }
              enum established {
```

```
                value 5;
                description
                  "Represents an open connection, data received can be
                   delivered to the user. The normal state for the data
                   transfer phase of the connection.";
              }
              enum fin-wait-1 {
                value 6;
                description
                  "Represents waiting for a connection termination
                   request from the remote TCP peer, or an
                   acknowledgment of the connection termination request
                   previously sent.";
              }
              enum fin-wait-2 {
                value 7;
                description
                  "Represents waiting for a connection termination
                   request from the remote TCP peer.";
              }
              enum close-wait {
                value 8;
                description
                  "Represents waiting for a connection termination
                   request from the local user.";
              }
              enum last-ack {
                value 9;
                description
                  "Represents waiting for an acknowledgment of the
                   connection termination request previously sent to
                   the remote TCP peer (this termination request sent
                   to the remote TCP peer already included an
                   acknowledgment of the termination request sent from
                   the remote TCP peer)";
              }
              enum closing {
                value 10;
                description
                  "Represents waiting for a connection termination
                   request acknowledgment from the remote TCP peer.";
              }
              enum time-wait {
                value 11;
                description
                  "Represents waiting for enough time to pass to be
                   sure the remote TCP peer received the acknowledgment
                   of its connection termination request, and to avoid
```

```
                    new connections being impacted by delayed segments
                    from previous connections.";
                }
              }
              config false;
              description
                "The state of this TCP connection.";
            }
            description
              "List of TCP connections with their parameters.

               The list is modeled as writeable even though only some of
               the nodes are writeable, e.g. keepalive. Connections
               that are created and match this list SHOULD apply the
               writeable parameters. At the same time, implementations
               may not allow creation of new TCP connections simply by
               adding entries to the list. Furthermore, the behavior
               upon removal is implementation-specific. Implementations
               may not support closing or resetting a TCP connection
               upon an operation that removes the entry from the list.

               The operational state of this list SHOULD reflect
               connections that have configured, but not created, and
               connections that have been created. Connections in
               CLOSED state are not reflected on this list.";
          }
          description
            "A container of all TCP connections.";
        }

        list tcp-listeners {
          key "type address port";
          config false;

          description
            "A table containing information about a particular
             TCP listener.";

          leaf type {
            type inet:ip-version;
            description
              "The address type of address.  The value
               should be unspecified (0) if connection initiations
               to all local IP addresses are accepted.";
          }

          leaf address {
            type union {
```

```
            type inet:ip-address;
            type string {
              length 0;
            }
          }
          description
            "The local IP address for this TCP connection.

             The value of this node can be represented in three
             possible ways, depending on the characteristics of the
             listening application:

             1. For an application willing to accept both IPv4 and
                IPv6 datagrams, the value of this node must be
                ''h (a zero-length octet-string), with the value
                of the corresponding 'type' object being
                unspecified (0).

             2. For an application willing to accept only IPv4 or
                IPv6 datagrams, the value of this node must be
                '0.0.0.0' or '::' respectively, with
                'type' representing the appropriate address type.

             3. For an application which is listening for data
                destined only to a specific IP address, the value
                of this node is the specific local address, with
                'type' representing the appropriate address type.";
        }

        leaf port {
          type inet:port-number;
          description
            "The local port number for this TCP connection.";
        }
      }

      container statistics {
        if-feature statistics;
        config false;

        leaf active-opens {
          type yang:counter64;
          description
            "The number of times that TCP connections have made a
             direct transition to the SYN-SENT state from the CLOSED
             state.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
```

```
                    Specification.";
        }

        leaf passive-opens {
          type yang:counter64;
          description
            "The number of times TCP connections have made a direct
             transition to the SYN-RCVD state from the LISTEN state.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf attempt-fails {
          type yang:counter64;
          description
            "The number of times that TCP connections have made a
             direct transition to the CLOSED state from either the
             SYN-SENT state or the SYN-RCVD state, plus the number of
             times that TCP connections have made a direct transition
             to the LISTEN state from the SYN-RCVD state.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf establish-resets {
          type yang:counter64;
          description
            "The number of times that TCP connections have made a
             direct transition to the CLOSED state from either the
             ESTABLISHED state or the CLOSE-WAIT state.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf currently-established {
          type yang:gauge32;
          description
            "The number of TCP connections for which the current state
             is either ESTABLISHED or CLOSE-WAIT.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf in-segments {
```

```
          type yang:counter64;
          description
            "The total number of TCP segments received, including those
             received in error. This count includes TCP segments
             received on currently established connections.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf out-segments {
          type yang:counter64;
          description
            "The total number of TCP segments sent, including those on
             current connections but excluding those containing only
             retransmitted octets.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf retransmitted-segments {
          type yang:counter64;
          description
            "The total number of TCP segments retransmitted; that is,
             the number of TCP segments transmitted containing one or
             more previously transmitted octets.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf in-errors {
          type yang:counter64;
          description
            "The total number of TCP segments received in error
             (e.g., bad TCP checksums).";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
                       Specification.";
        }

        leaf out-resets {
          type yang:counter64;
          description
            "The number of TCP segments sent containing the RST flag.";
          reference
            "RFC 9293: Transmission Control Protocol (TCP)
```

```
                        Specification.";
        }

        leaf auth-failures {
          if-feature authentication;
          type yang:counter64;
          description
            "The number of times that authentication has failed either
             with TCP-AO or MD5.";
        }

        action reset {
          nacm:default-deny-all;
          description
            "Reset statistics action command.";
          input {
            leaf reset-at {
              type yang:date-and-time;
              description
                "Time when the reset action needs to be
                 executed.";
            }
          }
          output {
            leaf reset-finished-at {
              type yang:date-and-time;
              description
                "Time when the reset action command completed.";
            }
          }
        }
        description
          "Statistics across all connections.";
      }
    }

    augment "/key-chain:key-chains/key-chain:key-chain/key-chain:key" {
      description
        "Augmentation of the key-chain model to add TCP-AO and TCP-MD5
         authentication.";

      container authentication {
        if-feature authentication;
        leaf keychain {
          type key-chain:key-chain-ref;
          description
            "Reference to the key chain that will be used by
             this model. Applicable for TCP-AO and TCP-MD5
```

```
             only";
           reference
             "RFC 8177: YANG Key Chain.";
         }

         choice authentication {
           container ao {
             presence "Presence container for all TCP-AO related" +
                      " configuration";
             uses ao;
             description
               "Use TCP-AO to secure the connection.";
           }

           container md5 {
             presence "Presence container for all MD5 related" +
                      " configuration";
             description
               "Use TCP-MD5 to secure the connection. As the TCP MD5
                signature option is obsoleted by TCP-AO, it is
                RECOMMENDED to use TCP-AO instead.";
             reference
               "RFC 2385: Protection of BGP Sessions via the TCP MD5
                          Signature.";
           }
           description
             "Choice of TCP authentication.";
         }
         description
           "Authentication definitions for TCP configuration.
            This includes parameters such as how to secure the
            connection, that can be part of either the client
            or server.";
       }
     }
   }
   <CODE ENDS>
```

5. IANA Considerations

5.1. The IETF XML Registry

This document registers an URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

The following entry is requested to be added to the "YANG Module Names" registry created by YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) [RFC6020]:

```
name:         ietf-tcp
namespace:    urn:ietf:params:xml:ns:yang:ietf-tcp
prefix:       tcp
reference:    RFC XXXX (this document)
```

The registration is not maintained by IANA.

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

* Common configuration included from NETCONF Client and Server Models [I-D.ietf-netconf-tcp-client-server]. Unrestricted access to all the nodes, e.g., keepalive idle-timer, can cause connections to fail or to timeout prematurely.

* Authentication configuration. Unrestricted access to the nodes under authentication configuration can prevent the use of authenticated communication and cause connection setups to fail. This can result in massive security vulnerabilities and service disruption for the traffic requiring authentication.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

* Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack.

* Similarly, unrestricted access to statistics of the client or server can be used by a malicious user to exploit any vulnerabilities of the system.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

* The YANG module allows for the statistics to be cleared by executing the reset action. This action should be restricted to users with the right permission.

The module specified in this document supports MD5 to basically accommodate the installed BGP base. MD5 suffers from the security weaknesses discussed in Section 2 of RFC 6151 [RFC6151] or Section 2.1 of RFC 6952 [RFC6952].

7. References

7.1. Normative References

[I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-13, 24 May 2022, <https://www.ietf.org/archive/id/draft-ietf-netconf-tcp-client-server-13.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, DOI 10.17487/RFC5926, June 2010, <https://www.rfc-editor.org/info/rfc5926>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <https://www.rfc-editor.org/info/rfc8177>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

7.2. Informative References

[I-D.ietf-i2nsf-capability-data-model]
Hares, S., Jeong, J. P., Kim, J. T., Moskowitz, R., and Q. Lin, "I2NSF Capability YANG Data Model", Work in Progress, Internet-Draft, draft-ietf-i2nsf-capability-data-model-32, 23 May 2022, <https://www.ietf.org/archive/id/draft-ietf-i2nsf-capability-data-model-32.txt>.

[I-D.ietf-i2nsf-nsf-facing-interface-dm]
Kim, J. T., Jeong, J. P., Park, J., Hares, S., and Q. Lin, "I2NSF Network Security Function-Facing Interface YANG Data Model", Work in Progress, Internet-Draft, draft-ietf-i2nsf-nsf-facing-interface-dm-29, 1 June 2022, <https://www.ietf.org/archive/id/draft-ietf-i2nsf-nsf-facing-interface-dm-29.txt>.

[I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-model-14, 3 July 2022, <https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-14.txt>.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P. S., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-interface-16, 31 August 2022, <https://www.ietf.org/archive/id/draft-ietf-taps-interface-16.txt>.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <https://www.rfc-editor.org/info/rfc4022>.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <https://www.rfc-editor.org/info/rfc4364>.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <https://www.rfc-editor.org/info/rfc4898>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <https://www.rfc-editor.org/info/rfc6643>.

[RFC6952] Jethanandani, M., Patel, K., and L. Zheng, "Analysis of BGP, LDP, PCEP, and MSDP Issues According to the Keying and Authentication for Routing Protocols (KARP) Design Guide", RFC 6952, DOI 10.17487/RFC6952, May 2013, <https://www.rfc-editor.org/info/rfc6952>.

[RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <https://www.rfc-editor.org/info/rfc8512>.

[RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <https://www.rfc-editor.org/info/rfc8513>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

[RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <https://www.rfc-editor.org/info/rfc8783>.

[RFC9182] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., Munoz, L., and A. Aguado, "A YANG Network Data Model for Layer 3 VPNs", RFC 9182, DOI 10.17487/RFC9182, February 2022, <https://www.rfc-editor.org/info/rfc9182>.

[RFC9235] Touch, J. and J. Kuusisaari, "TCP Authentication Option (TCP-AO) Test Vectors", RFC 9235, DOI 10.17487/RFC9235, May 2022, <https://www.rfc-editor.org/info/rfc9235>.

Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews (in alphabetical order): Mohamed Boucadair, Gorry Fairhurst, Jeffrey Haas, and Tom Petch.

Appendix B. Examples

B.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

NOTE: '\' line wrapping per RFC 8792

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example shows how TCP keepalive, MSS and PMTU can be configured \

for a given connection. An idle connection is dropped after
idle-time + (max-probes * probe-interval).
-->
<tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
  <connections>
    <connection>
      <local-address>192.0.2.1</local-address>
      <remote-address>192.0.2.2</remote-address>
      <local-port>1025</local-port>
      <remote-port>22</remote-port>
      <mss>1400</mss>
      <pmtud>true</pmtud>
      <keepalives>
        <idle-time>5</idle-time>
        <max-probes>5</max-probes>
        <probe-interval>10</probe-interval>
      </keepalives>
    </connection>
  </connections>
</tcp>
```

B.2. TCP-AO Configuration

The following example demonstrates how to model a TCP-AO [RFC5925] configuration for the example in TCP-AO Test Vectors [RFC9235]. The IP addresses and other parameters are taken from the test vectors.

NOTE: '\' line wrapping per RFC 8792

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example sets TCP-AO configuration parameters similar to
the examples in RFC 9235.
-->

<key-chains
    xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
  <key-chain>
    <name>ao-config</name>
    <description>"An example for TCP-AO configuration."</description>\

    <key>
      <key-id>55</key-id>
      <lifetime>
        <send-lifetime>
          <start-date-time>2017-01-01T00:00:00Z</start-date-time>
          <end-date-time>2017-02-01T00:00:00Z</end-date-time>
        </send-lifetime>
        <accept-lifetime>
          <start-date-time>2016-12-31T23:59:55Z</start-date-time>
          <end-date-time>2017-02-01T00:00:05Z</end-date-time>
        </accept-lifetime>
      </lifetime>
      <crypto-algorithm
          xmlns:tcp=
          "urn:ietf:params:xml:ns:yang:ietf-tcp">tcp:aes-128</crypto-algorit\
hm>
      <key-string>
        <keystring>testvector</keystring>
      </key-string>
      <authentication
          xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
        <keychain>ao-config</keychain>
        <ao>
          <send-id>61</send-id>
          <recv-id>84</recv-id>
        </ao>
      </authentication>
    </key>
  </key-chain>
</key-chains>
```

Appendix C. Complete Tree Diagram

Here is the complete tree diagram for the TCP YANG model.

```
module: ietf-tcp
  +--rw tcp!
     +--rw connections
     |  +--rw connection*
     |          [local-address remote-address local-port remote-port]
     |     +--rw local-address     inet:ip-address
     |     +--rw remote-address    inet:ip-address
     |     +--rw local-port        inet:port-number
     |     +--rw remote-port       inet:port-number
     |     +--rw mss?              mss
     |     +--rw pmtud?            boolean
     |     +--rw keepalives! {keepalives-supported}?
     |     |  +--rw idle-time         uint16
     |     |  +--rw max-probes        uint16
     |     |  +--rw probe-interval    uint16
     |     +--ro state?            enumeration
     +--ro tcp-listeners* [type address port]
     |  +--ro type       inet:ip-version
     |  +--ro address    union
     |  +--ro port       inet:port-number
     +--ro statistics {statistics}?
        +--ro active-opens?             yang:counter64
        +--ro passive-opens?            yang:counter64
        +--ro attempt-fails?            yang:counter64
        +--ro establish-resets?         yang:counter64
        +--ro currently-established?    yang:gauge32
        +--ro in-segments?              yang:counter64
        +--ro out-segments?             yang:counter64
        +--ro retransmitted-segments?   yang:counter64
        +--ro in-errors?                yang:counter64
        +--ro out-resets?               yang:counter64
        +--ro auth-failures?            yang:counter64
        |       {authentication}?
        +---x reset
           +---w input
           |  +---w reset-at?   yang:date-and-time
           +--ro output
              +--ro reset-finished-at?   yang:date-and-time

  augment /key-chain:key-chains/key-chain:key-chain/key-chain:key:
    +--rw authentication {authentication}?
       +--rw keychain?    key-chain:key-chain-ref
       +--rw (authentication)?
          +--:(ao)
```

```
             |  +--rw ao!
             |     +--rw send-id?               uint8
             |     +--rw recv-id?               uint8
             |     +--rw include-tcp-options?   boolean
             |     +--rw accept-key-mismatch?   boolean
             |     +--ro r-next-key-id?         uint8
             +--:(md5)
                +--rw md5!
```

Authors' Addresses

Michael Scharf
Hochschule Esslingen - University of Applied Sciences
Kanalstr. 33
73728 Esslingen
Germany
Email: michael.scharf@hs-esslingen.de

Mahesh Jethanandani
Kloud Services
Email: mjethanandani@gmail.com

Vishal Murgai
Samsung
Email: vmurgai@gmail.com

TCPM WG
Internet Draft
Intended status: Experimental
Expires: April 2022

J. Touch
Independent consultant
T. Faber
The Aerospace Corporation

October 12, 2021

TCP SYN Extended Option Space Using an Out-of-Band Segment
draft-touch-tcpm-tcp-syn-ext-opt-10.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 12, 2019.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document describes an experimental method to extend the option space for connection parameters within the initial TCP SYN segment, at the start of a TCP connection. This method effectively extends the option space of an initial SYN by using an additional coupled segment that is sent 'out-of-band'. It complements the proposed Extended Data Offset (EDO) option that is applicable only after the initial segment.

Table of Contents

1. Introduction....2
2. Conventions used in this document....3
3. Experiment Goals....3
4. Using Multiple Segments to Establish a Connection....4
5. The TCP SYN-EOS Option....5
5.1. Reliable Delivery of Lone Initial Segments....7
5.2. Reliable Delivery of a Lone SYN with SYN-EOS....7
5.3. Interaction with EDO....8
6. Issues....9
6.1. General Issues....9
6.2. Option processing order....9
6.3. Middlebox Transit Issues....10
6.4. Interaction with Other TCP Options....11
6.5. TCP Fast Open....11
6.5.1. TCP Authentication Option and TCP MD5....11
7. TCP SYN-EOS Interaction with TCP....11
7.1. TCP User Interface....11
7.2. TCP States and Transitions....11
7.3. TCP Segment Processing....11
7.4. Impact on TCP Header Size....11
8. Error Conditions....12
8.1. Connectionless Resets....12
8.2. ICMP Handling....12
9. Security Considerations....12
10. IANA Considerations....12
11. References....12
11.1. Normative References....12
11.2. Informative References....12
12. Acknowledgments....13

1. Introduction

This document describes a method to extend the option space available in the initial SYN segment of a TCP connection (e.g., SYN

set and ACK not set) [RFC793]. This extension is required to support some combinations of TCP options, notably large ones such as TCP AO [RFC5925], Multipath TCP [RFC6824], and TCP Fast Open [RFC7413] with other options already typically used in most TCP connections. This document specifies this TCP SYN extended option space (SYN-EOS) option and is independent of (and thus compatible with) IPv4 and IPv6. SYN-EOS complements the proposed TCP Extended Data Offset (EDO) option, which increases the space available for options in all segments except the initial SYN [To18].

2. Conventions used in this document

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 RFC-2119 [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.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that initiates a connection using two coupled segments instead of the traditional single one. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness, and interoperability with itself, with other variants of TCP and with common network equipment, whether standardized or not.

The following describe the criteria that define success for this experiment and its expected duration.

Success criteria: The experimental protocol will be considered successful if, in the consensus opinion of the IETF, it functions correctly in a sufficiently wide scope to be useful and it does no harm, which implies that it ought to introduce minimal additional delay or load to either updated or existing implementations and it introduces no new security vulnerabilities. It is also required not

to be unduly difficult or complex to implement correctly, so that it is not likely to lead to additional bugs or vulnerabilities.

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

4. Using Multiple Segments to Establish a Connection

The basis of SYN-EOS is the use of multiple TCP segments to initiate a TCP connection. It is also possible to extend initial SYN option space using context established from prior connections or using separate TCP connections (e.g., using the FTP control channel), this document focuses on a mechanisms that applies to any connection (including the first between two hosts) and do not require prior or established concurrent TCP connections.

There are four examples of such approaches:

- Send a primary SYN and an extension SYN (LOIC [Yo11])
- Send a primary SYN and extension non-SYN data (LO/SLO [Ed08])
- Send two separate SYNs: a legacy SYN and an upgraded SYN on separate port pairs (dual-stack, named "Sister-SYN" [Br14])
- Send a primary SYN and an extension out-of-band segment (OOB, this document)

All four approaches extend the space available in the initial SYN by sending an additional segment during the first phase of the three-way handshake. The Long Options by Invalid Checksum (LOIC) approach differentiates the two SYNs by using an invalid TCP checksum in the extension SYN [Yo11], which thus cannot traverse NAT/NAPT devices [RFC3234].

The LO/SLO approach extends the three-way handshake into a five-way handshake to include extra options during the third segment, so the traditional SYN/ACK does not complete the active connection [Ed08]. In current implementations, the client TCP state machine transitions to the ESTABLISHED state upon receipt of the SYN/ACK (including transmission of the resulting ACK). In SLO, additional options sent during the third segment are treated as part of the initial SYN and the fourth segment with responses to these options is treated as part of the conventional SYN/ACK. As with conventional TCP, data can

be sent during this handshake as part of any segment, but this data needs to wait for the entire handshake to complete before being forwarded to the application to ensure that all options have been negotiated successfully. This adds an additional round trip of latency which is undesirable in many cases. Connection-splitting middleboxes that merge these segments might also cause long options to be interpreted as data.

The Sister-SYN approach is a dual-stack mechanism. Both legacy servers and upgraded servers process both SYNs; clients terminate the appropriate pending connection based on whether the option is acknowledged for each connection.

The remainder of this document presents the SYN-EOS approach, which overcomes these limitations using an out-of-band segment to extend the option space of the SYN.

5. The TCP SYN-EOS Option

The SYN-EOS approach uses a primary conventional SYN and an additional out-of-band data segment, the latter being a non-SYN packet with the ACK flag not set. Additional options are placed in payload of an out-of-band (OOB) segment, i.e., a segment whose ACK bit is cleared but is not a SYN (i.e., both SYN and ACK are zero). This offers the following advantages:

1. It provide expansion space for options on a SYN, limited only by the default maximum segment size (535 payload bytes for IPv4);

2. It reduces the chances that middleboxes will alter the extra options, given there is a higher bar to altering the payload than header fields.

3. It allows for future structured ways to hide extra options from middleboxes and/or to protect them from being altered.

A client initiating a TCP connection (i.e., issuing an active open) uses the SYN-EOS option flag to indicate the presence of the extended option space (Figure 1). This follows the TCP option format, where Kind is SYN-EOS-OPT and Length is 2.

```
+--------+--------+
|  Kind  | Length |
+--------+--------+
```

Figure 1 TCP SYN-EOS OOB option

An upgraded client supporting this feature uses this option only when the space needed for options in the initial SYN exceeds that of legacy TCP. When needed, the client sends the SYN-EOS option in the initial SYN, together with whatever other options are intended for connections to legacy servers (i.e., passive listeners). A legacy server would respond with a SYN/ACK without the SYN-EOS option, while also confirming other supported options, and the connection would proceed without the SYN-EOS extension.

The upgraded client that sends the initial SYN using this option also sends an out-of-band (OOB) data segment with the same option to the same source and destination addresses and ports as the initial SYN. An OOB data segment is herein defined as a TCP segment in which neither SYN nor ACK flag is set. In particular, this looks like a conventional data segment with the ACK field cleared. Current TCP requirements allow the ACK field to be cleared for only the initial SYN, so this segment looks like a data segment that has been transmitted 'out-of-band', before a connection has been established. The entire payload of the segment is used for additional options.

Upgraded servers that receive the TCP SYN with the SYN-EOS option wait for the corresponding OOB segment and treat the entire set of options in both segments as if they arrived with the initial SYN. Once both have arrived, the server first processes TCP options placed before each SYN-EOS option, applying them solely to their own individual segment. Then the server marshals together all the TCP options placed after each SYN-EOS option. It applies them to the initial SYN only, as if they had all been concatenated after the SYN-EOS OOB option.

>> The server MUST process the options placed after each SYN-EOS option in the following order:

1. Those in the option space of the initial SYN

2. Those in the option space of the OOB segment

3. Those in the payload space of the OOB segment

The upgraded server proceeds with the remainder of the connection as if the SYN-EOS OOB option were a also EDO request option [To18] in the SYN.

>> The SYN/ACK MUST include the SYN-EOS option to confirm the server's support for both the SUN-EOS and EDO capabilities and to confirm receipt of both the SYN and OOB segment. The server MAY also

extend the options space of the SYN/ACK using the EDO option if needed.

>> Any host that supports SYN-EOS MUST also thus support EDO.

>> The OOB segment MUST use the same sequence number as the initial SYN.

>> The client MUST NOT send multiple different OOB segments. If the server receives more than one OOB segment for the same connection it MUST solely use the first.

5.1. Reliable Delivery of Lone Initial Segments

The server acknowledges the initial segment and the OOB segment together by using a SYN/ACK that carries the appropriate SYN-EOS option. The following subsections describe how the server acknowledges initial segments after a certain time if only one has arrived.

5.2. Reliable Delivery of a Lone SYN with SYN-EOS

If an upgraded server has received only a SYN with the SYN-EOS option but no corresponding OOB segment, after a certain time it MUST proceed with the connection as if the SYN had been received without the SYN-EOS option. I.e. it processes all other TCP options and responds with a SYN/ACK without the SYN-EOS option.

A client will not be able to tell whether this SYN/ACK is from a legacy server or an upgraded server. How the client proceeds on receipt of such a SYN/ACK depends on whether it wishes to retry sending the TCP options in the OOB segment or to proceed without them (e.g. for latency reasons):

>> If the client chooses to proceed without the OOB segment, it MUST proceed as if the SYN-EOS option had never been used, by sending an ACK to complete the three-way handshake.

>> If the client chooses to retry, it MUST retransmit the OOB segment with the same sequence number as the ISN of the SYN, so it is still out-of-band. However, this time it sets the ACK flag and it sets the acknowledgement number to one greater than the sequence number of the SYN/ACK. This effectively acknowledges receipt of the SYN/ACK, but requests a fuller SYN/ACK that also covers the OOB segment. At this stage a client that has chosen to retry the OOB segment MUST NOT send the ACK that would normally complete the three-way handshake.

>> If an upgraded server receives such a retransmitted OOB segment, it MUST process the additional TCP options as if they were placed after those in the initial SYN. Then it MUST send a SYN/ACK containing the SYN-EOS option, as if it had not sent the earlier SYN/ACK .

On receipt of this SYN/ACK, the client sends an ACK to complete the handshake.

>> If an upgraded server receives an ACK to complete the handshake, then later receives an OOB segment, it MUST discard the late OOB segment.

>> If a server, whether upgraded or not, receives only an OOB segment and no corresponding SYN, it MUST discard it and it MUST NOT ever respond (see Security Considerations).

5.3. Interaction with EDO

Successful negotiation of either SYN-EOS option has the same effect as EDO. Successful SYN-EOS negotiation enables EDO for the remainder of the connection.

>> After successful SYN-EOS negotiation, segments after the initial SYN MAY use the EDO option.

Note that a failure to negotiate SYN-EOS has also fails to automatically negotiate EDO for endpoints that support EDO but not SYN-EOS. As a consequence:

>> If EDO is desired when SYN-EOS fails, the initial SYN options MUST include a separate EDO Supported option.

If SYN-EOS is sent in the initial SYN and confirmed in the SYN/ACK, EDO is available for the remainder of the connection. Segments that need to extend their option space would then include EDO.

>> If SYN-EOS and EDO Supported are sent in the initial SYN and received by SYN-EOS capable server, the server MUST include SYN-EOS in the SYN/ACK, and MAY also include EDO Extension if needed to provide additional option space.

>> If the server agrees to EDO but cannot support SYN-EOS, the SYN/ACK MUST include EDO Supported as per [To18] to confirm the capability.

6. Issues

The following issues are known.

6.1. General Issues

Caching is required because it is unlikely that both segments involved in initiating a SYN-EOS connection will arrive at the same time:

>> Servers supporting SYN-EOS SHOULD cache received initial SYNs with the SYN-EOS option. Servers MAY decline to cache received initial SYNs if they are under memory constraints.

>> Servers supporting SYN-EOS SHOULD cache received SYN-C segments with the SYN-EOS option. Servers MAY cache received OOB segments but MUST NOT examine or process them further in any way until their corresponding SYN segment arrives.

Similarly, clients need to be able to retransmit supplements to ensure their delivery:

>> Clients MUST retransmit the supplemental segment any time they retransmit the initial SYN segment.

Should this be a new option or just a variant of EDO, and if so, how would it change EDO?

SYN Cookies: An updated server can achieve the same outcome as SYN cookies by putting all the necessary connection state in TCP options in the SYN/ACK (using EDO if extra space is needed). It would then discard its own copy of this state, which it could recover from the TCP options in the final ACK of the 3WHS sent by the client. New TCP options complementary to SYN-EOS might need to be defined to achieve this for some types of TCP option (TBA). A legacy server will not understand the SYN-EOS option whether it uses SYN cookies or not, so it will provide the same legacy service whether or not it uses SYN cookies.

Useful to send SYN, wait shortly, then send OOB

OOB traversal concerns

6.2. Option processing order

TCP options before the EOS-SYN on initial SYN segments are necessarily processed individually when each segment arrives. When

both segments of an EOS-SYN connection establishment arrive, the remaining options are processed in the following sequence:

1. Initial SYN options

2. Supplement options

3. Supplement payload

*** NOTE TO THE WG:

There are two other constraints that might be applied to the supplement options:

>> I. Supplement options MUST exactly match initial SYN options.

>> II. Supplement options MUST contain only the SYN-EOS option.

If either of these is chosen, the supplement options are NOT processed again (i.e., they are discarded).

The former constraint helps the supplement segment share the same fate as the initial SYN. The latter recognizes that the supplement option space is not needed given the supplement payload, because the option space is created from the payload space anyway.

*** END NOTE

6.3. Middlebox Transit Issues

NB: this variant will require an additional 1-byte field on the SYN-EOS option for the EOO field.

Traversal of middleboxes that ensure the payload matches the destination port number. It would be possible to include the facility for SYN-EOS to include an Extra Option Offset (EOO) field. A client setting EOO to a non-zero value would offset the start of the additional TCP options by this number of 4-byte words from the start of the payload.

>> An upgraded SYN-EOS server MUST start reading the additional TCP options from a point within the payload that is offset by this number of 4-byte words from the start of the payload. An upgraded SYN-EOS server MUST ignore all data in the payload up to this point.

The client would then be free to include fake data at the start of the payload consistent with what a middlebox might expect for the

destination port in use. The data to use would be application and implementation dependent and is not determined in the present specification.

6.4. Interaction with Other TCP Options

6.5. TCP Fast Open

TBD.

Notes: SYN-EOS appears to be safe with TFO. Dual-SYN variants appear to have potential problems with both upgraded and legacy servers. With upgraded servers, receipt of a legacy SYN with the SYN extension option flag present might require delayed response. With legacy servers, it may be impossible to safely use TFO with the extended SYN.

6.5.1. TCP Authentication Option and TCP MD5

TBD.

Notes: Likely to be similar to TCP EDO, i.e., requiring authentication processing before extension processing.

7. TCP SYN-EOS Interaction with TCP

The following subsections describe how SYN-EOS interacts with the TCP specification [RFC793].

7.1. TCP User Interface

TBD.

7.2. TCP States and Transitions

TBD.

7.3. TCP Segment Processing

TBD.

7.4. Impact on TCP Header Size

TBD.

8. Error Conditions

8.1. Connectionless Resets

TBD.

8.2. ICMP Handling

TBD [RFC792].

9. Security Considerations

>> By default, a SYN-EOS server must not cache an OOB segment and MUST NOT respond to an OOB segment if it arrives before the corresponding SYN segment, because many legacy firewalls will allow OOB segments into private networks. Caching of OOB segments MAY be enabled explicitly on public servers.

More TBD.

10. IANA Considerations

TBD.

This section is to be removed prior to publication as an RFC.

11. References

11.1. Normative References

[RFC793] 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.

[To21] Touch, J., W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo (work in progress), Nov. 2021.

11.2. Informative References

[Br14] Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

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

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

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

[Yo11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

12. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback.

The use of multiple segments to extend the option space of a SYN was initially proposed by Bob Briscoe. His initial proposal used complementary SYNs in an earlier version of this document, which evolved into mutually-exclusive "Sister-SYNs" in [Br14].

This work is partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors’ Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Ted Faber
Engineering Specialist
Computer Systems Research Department
The Aerospace Corporation
2310 E. El Segundo Blvd.
El Segundo, CA 90245-4609 USA

Phone: +1 (310) 336-7373
Email: theodore.v.faber@aero.org

TCPM WG
Internet Draft
Intended status: Experimental
Expires: April 2023

J. Touch
Independent Consultant
T. Faber
The Aerospace Corporation

October 22, 2022

TCP SYN Extended Option Space Using an Out-of-Band Segment
draft-touch-tcpm-tcp-syn-ext-opt-12.txt

Abstract

This document describes an experimental method to extend the option space for connection parameters within the initial TCP SYN segment, at the start of a TCP connection. This method effectively extends the option space of an initial SYN by using an additional coupled segment that is sent 'out-of-band'. It complements the proposed Extended Data Offset (EDO) option that is applicable only after the initial segment.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

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 22, 2019.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction..... 3
2. Conventions used in this document..... 3
3. Experiment Goals..... 3
4. Using Multiple Segments to Establish a Connection..... 4
5. The TCP SYN-EOS Option..... 5
   5.1. Reliable Delivery of Lone Initial Segments..... 7
   5.2. Reliable Delivery of a Lone SYN with SYN-EOS..... 7
   5.3. Interaction with EDO..... 8
6. Issues..... 9
   6.1. General Issues..... 9
   6.2. Option processing order..... 10
   6.3. Middlebox Transit Issues..... 10
   6.4. Interaction with Other TCP Options..... 11
   6.5. TCP Fast Open..... 11
      6.5.1. TCP Authentication Option and TCP MD5..... 11
7. TCP SYN-EOS Interaction with TCP..... 11
   7.1. TCP User Interface..... 11
   7.2. TCP States and Transitions..... 11
   7.3. TCP Segment Processing..... 11
   7.4. Impact on TCP Header Size..... 12
8. Error Conditions..... 12
   8.1. Connectionless Resets..... 12
   8.2. ICMP Handling..... 12
9. Security Considerations..... 12
10. IANA Considerations..... 12
11. References..... 12
   11.1. Normative References..... 12
   11.2. Informative References..... 13
12. Acknowledgments..... 13

1. Introduction

This document describes a method to extend the option space available in the initial SYN segment of a TCP connection (e.g., SYN set and ACK not set) [RFC9293]. This extension is required to support some combinations of TCP options, notably large ones such as TCP AO [RFC5925], Multipath TCP [RFC8684], and TCP Fast Open [RFC7413] with other options already typically used in most TCP connections. This document specifies this TCP SYN extended option space (SYN-EOS) option and is independent of (and thus compatible with) IPv4 and IPv6. SYN-EOS complements the proposed TCP Extended Data Offset (EDO) option, which increases the space available for options in all segments except the initial SYN [To22].

2. Conventions used in this document

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 RFC-2119 [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.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that initiates a connection using two coupled segments instead of the traditional single one. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness, and interoperability with itself, with other variants of TCP and with common network equipment, whether standardized or not.

The following describe the criteria that define success for this experiment and its expected duration.

Success criteria: The experimental protocol will be considered successful if, in the consensus opinion of the IETF, it functions correctly in a sufficiently wide scope to be useful and it does no

harm, which implies that it ought to introduce minimal additional delay or load to either updated or existing implementations and it introduces no new security vulnerabilities. It is also required not to be unduly difficult or complex to implement correctly, so that it is not likely to lead to additional bugs or vulnerabilities.

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

4. Using Multiple Segments to Establish a Connection

The basis of SYN-EOS is the use of multiple TCP segments to initiate a TCP connection. It is also possible to extend initial SYN option space using context established from prior connections or using separate TCP connections (e.g., using the FTP control channel), this document focuses on a mechanisms that applies to any connection (including the first between two hosts) and do not require prior or established concurrent TCP connections.

There are four examples of such approaches:

- Send a primary SYN and an extension SYN (LOIC [Yo11])
- Send a primary SYN and extension non-SYN data (LO/SLO [Ed08])
- Send two separate SYNs: a legacy SYN and an upgraded SYN on separate port pairs (dual-stack, named "Sister-SYN" [Br14])
- Send a primary SYN and an extension out-of-band segment (OOB, this document)

All four approaches extend the space available in the initial SYN by sending an additional segment during the first phase of the three-way handshake. The Long Options by Invalid Checksum (LOIC) approach differentiates the two SYNs by using an invalid TCP checksum in the extension SYN [Yo11], which thus cannot traverse NAT/NAPT devices [RFC3234].

The LO/SLO approach extends the three-way handshake into a five-way handshake to include extra options during the third segment, so the traditional SYN/ACK does not complete the active connection [Ed08]. In current implementations, the client TCP state machine transitions to the ESTABLISHED state upon receipt of the SYN/ACK (including transmission of the resulting ACK). In SLO, additional options sent

during the third segment are treated as part of the initial SYN and the fourth segment with responses to these options is treated as part of the conventional SYN/ACK. As with conventional TCP, data can be sent during this handshake as part of any segment, but this data needs to wait for the entire handshake to complete before being forwarded to the application to ensure that all options have been negotiated successfully. This adds an additional round trip of latency which is undesirable in many cases. Connection-splitting middleboxes that merge these segments might also cause long options to be interpreted as data.

The Sister-SYN approach is a dual-stack mechanism. Both legacy servers and upgraded servers process both SYNs; clients terminate the appropriate pending connection based on whether the option is acknowledged for each connection.

The remainder of this document presents the SYN-EOS approach, which overcomes these limitations using an out-of-band segment to extend the option space of the SYN.

5. The TCP SYN-EOS Option

The SYN-EOS approach uses a primary conventional SYN and an additional out-of-band data segment, the latter being a non-SYN packet with the ACK flag not set. Additional options are placed in payload of an out-of-band (OOB) segment, i.e., a segment whose ACK bit is cleared but is not a SYN (i.e., both SYN and ACK are zero). This offers the following advantages:

1. It provide expansion space for options on a SYN, limited only by the default maximum segment size (535 payload bytes for IPv4);

2. It reduces the chances that middleboxes will alter the extra options, given there is a higher bar to altering the payload than header fields.

3. It allows for future structured ways to hide extra options from middleboxes and/or to protect them from being altered.

A client initiating a TCP connection (i.e., issuing an active open) uses the SYN-EOS option flag to indicate the presence of the extended option space (Figure 1). This follows the TCP option format, where Kind is SYN-EOS-OPT and Length is 2.

```
+--------+--------+
|  Kind  | Length |
+--------+--------+
```

Figure 1 TCP SYN-EOS OOB option

An upgraded client supporting this feature uses this option only when the space needed for options in the initial SYN exceeds that of legacy TCP. When needed, the client sends the SYN-EOS option in the initial SYN, together with whatever other options are intended for connections to legacy servers (i.e., passive listeners). A legacy server would respond with a SYN/ACK without the SYN-EOS option, while also confirming other supported options, and the connection would proceed without the SYN-EOS extension.

The upgraded client that sends the initial SYN using this option also sends an out-of-band (OOB) data segment with the same option to the same source and destination addresses and ports as the initial SYN. An OOB data segment is herein defined as a TCP segment in which neither SYN nor ACK flag is set. In particular, this looks like a conventional data segment with the ACK field cleared. Current TCP requirements allow the ACK field to be cleared for only the initial SYN, so this segment looks like a data segment that has been transmitted 'out-of-band', before a connection has been established. The entire payload of the segment is used for additional options.

Upgraded servers that receive the TCP SYN with the SYN-EOS option wait for the corresponding OOB segment and treat the entire set of options in both segments as if they arrived with the initial SYN. Once both have arrived, the server first processes TCP options placed before each SYN-EOS option, applying them solely to their own individual segment. Then the server marshals together all the TCP options placed after each SYN-EOS option. It applies them to the initial SYN only, as if they had all been concatenated after the SYN-EOS OOB option.

>> The server MUST process the options placed after each SYN-EOS option in the following order:

1. Those in the option space of the initial SYN

2. Those in the option space of the OOB segment

3. Those in the payload space of the OOB segment

The upgraded server proceeds with the remainder of the connection as if the SYN-EOS OOB option were a also EDO request option [To22] in the SYN.

>> The SYN/ACK MUST include the SYN-EOS option to confirm the server's support for both the SUN-EOS and EDO capabilities and to confirm receipt of both the SYN and OOB segment. The server MAY also extend the options space of the SYN/ACK using the EDO option if needed.

>> Any host that supports SYN-EOS MUST also thus support EDO.

>> The OOB segment MUST use the same sequence number as the initial SYN.

>> The client MUST NOT send multiple different OOB segments. If the server receives more than one OOB segment for the same connection it MUST solely use the first.

5.1. Reliable Delivery of Lone Initial Segments

The server acknowledges the initial segment and the OOB segment together by using a SYN/ACK that carries the appropriate SYN-EOS option. The following subsections describe how the server acknowledges initial segments after a certain time if only one has arrived.

5.2. Reliable Delivery of a Lone SYN with SYN-EOS

If an upgraded server has received only a SYN with the SYN-EOS option but no corresponding OOB segment, after a certain time it MUST proceed with the connection as if the SYN had been received without the SYN-EOS option. I.e. it processes all other TCP options and responds with a SYN/ACK without the SYN-EOS option.

A client will not be able to tell whether this SYN/ACK is from a legacy server or an upgraded server. How the client proceeds on receipt of such a SYN/ACK depends on whether it wishes to retry sending the TCP options in the OOB segment or to proceed without them (e.g. for latency reasons):

>> If the client chooses to proceed without the OOB segment, it MUST proceed as if the SYN-EOS option had never been used, by sending an ACK to complete the three-way handshake.

>> If the client chooses to retry, it MUST retransmit the OOB segment with the same sequence number as the ISN of the SYN, so it

is still out-of-band. However, this time it sets the ACK flag and it sets the acknowledgement number to one greater than the sequence number of the SYN/ACK. This effectively acknowledges receipt of the SYN/ACK, but requests a fuller SYN/ACK that also covers the OOB segment. At this stage a client that has chosen to retry the OOB segment MUST NOT send the ACK that would normally complete the three-way handshake.

>> If an upgraded server receives such a retransmitted OOB segment, it MUST process the additional TCP options as if they were placed after those in the initial SYN. Then it MUST send a SYN/ACK containing the SYN-EOS option, as if it had not sent the earlier SYN/ACK .

On receipt of this SYN/ACK, the client sends an ACK to complete the handshake.

>> If an upgraded server receives an ACK to complete the handshake, then later receives an OOB segment, it MUST discard the late OOB segment.

>> If a server, whether upgraded or not, receives only an OOB segment and no corresponding SYN, it MUST discard it and it MUST NOT ever respond (see Security Considerations).

5.3. Interaction with EDO

Successful negotiation of either SYN-EOS option has the same effect as EDO. Successful SYN-EOS negotiation enables EDO for the remainder of the connection.

>> After successful SYN-EOS negotiation, segments after the initial SYN MAY use the EDO option.

Note that a failure to negotiate SYN-EOS has also fails to automatically negotiate EDO for endpoints that support EDO but not SYN-EOS. As a consequence:

>> If EDO is desired when SYN-EOS fails, the initial SYN options MUST include a separate EDO Supported option.

If SYN-EOS is sent in the initial SYN and confirmed in the SYN/ACK, EDO is available for the remainder of the connection. Segments that need to extend their option space would then include EDO.

>> If SYN-EOS and EDO Supported are sent in the initial SYN and received by SYN-EOS capable server, the server MUST include SYN-EOS

in the SYN/ACK, and MAY also include EDO Extension if needed to provide additional option space.

>> If the server agrees to EDO but cannot support SYN-EOS, the SYN/ACK MUST include EDO Supported as per [To22] to confirm the capability.

6. Issues

The following issues are known.

6.1. General Issues

Caching is required because it is unlikely that both segments involved in initiating a SYN-EOS connection will arrive at the same time:

>> Servers supporting SYN-EOS SHOULD cache received initial SYNs with the SYN-EOS option. Servers MAY decline to cache received initial SYNs if they are under memory constraints.

>> Servers supporting SYN-EOS SHOULD cache received SYN-C segments with the SYN-EOS option. Servers MAY cache received OOB segments but MUST NOT examine or process them further in any way until their corresponding SYN segment arrives.

Similarly, clients need to be able to retransmit supplements to ensure their delivery:

>> Clients MUST retransmit the supplemental segment any time they retransmit the initial SYN segment.

Should this be a new option or just a variant of EDO, and if so, how would it change EDO?

SYN Cookies: An updated server can achieve the same outcome as SYN cookies by putting all the necessary connection state in TCP options in the SYN/ACK (using EDO if extra space is needed). It would then discard its own copy of this state, which it could recover from the TCP options in the final ACK of the 3WHS sent by the client. New TCP options complementary to SYN-EOS might need to be defined to achieve this for some types of TCP option (TBA). A legacy server will not understand the SYN-EOS option whether it uses SYN cookies or not, so it will provide the same legacy service whether or not it uses SYN cookies.

Useful to send SYN, wait shortly, then send OOB

OOB traversal concerns

6.2. Option processing order

TCP options before the EOS-SYN on initial SYN segments are necessarily processed individually when each segment arrives. When both segments of an EOS-SYN connection establishment arrive, the remaining options are processed in the following sequence:

1. Initial SYN options

2. Supplement options

3. Supplement payload

*** NOTE TO THE WG:

There are two other constraints that might be applied to the supplement options:

>> I. Supplement options MUST exactly match initial SYN options.

>> II. Supplement options MUST contain only the SYN-EOS option.

If either of these is chosen, the supplement options are NOT processed again (i.e., they are discarded).

The former constraint helps the supplement segment share the same fate as the initial SYN. The latter recognizes that the supplement option space is not needed given the supplement payload, because the option space is created from the payload space anyway.

*** END NOTE

6.3. Middlebox Transit Issues

NB: this variant will require an additional 1-byte field on the SYN-EOS option for the EOO field.

Traversal of middleboxes that ensure the payload matches the destination port number. It would be possible to include the facility for SYN-EOS to include an Extra Option Offset (EOO) field. A client setting EOO to a non-zero value would offset the start of the additional TCP options by this number of 4-byte words from the start of the payload.

>> An upgraded SYN-EOS server MUST start reading the additional TCP options from a point within the payload that is offset by this number of 4-byte words from the start of the payload. An upgraded SYN-EOS server MUST ignore all data in the payload up to this point.

The client would then be free to include fake data at the start of the payload consistent with what a middlebox might expect for the destination port in use. The data to use would be application and implementation dependent and is not determined in the present specification.

6.4. Interaction with Other TCP Options

6.5. TCP Fast Open

TBD.

Notes: SYN-EOS appears to be safe with TFO. Dual-SYN variants appear to have potential problems with both upgraded and legacy servers. With upgraded servers, receipt of a legacy SYN with the SYN extension option flag present might require delayed response. With legacy servers, it may be impossible to safely use TFO with the extended SYN.

6.5.1. TCP Authentication Option and TCP MD5

TBD.

Notes: Likely to be similar to TCP EDO, i.e., requiring authentication processing before extension processing.

7. TCP SYN-EOS Interaction with TCP

The following subsections describe how SYN-EOS interacts with the TCP specification [RFC9293].

7.1. TCP User Interface

TBD.

7.2. TCP States and Transitions

TBD.

7.3. TCP Segment Processing

TBD.

7.4. Impact on TCP Header Size

TBD.

8. Error Conditions

8.1. Connectionless Resets

TBD.

8.2. ICMP Handling

TBD [RFC792].

9. Security Considerations

>> By default, a SYN-EOS server must not cache an OOB segment and MUST NOT respond to an OOB segment if it arrives before the corresponding SYN segment, because many legacy firewalls will allow OOB segments into private networks. Caching of OOB segments MAY be enabled explicitly on public servers.

More TBD.

10. IANA Considerations

TBD.

This section is to be removed prior to publication as an RFC.

11. References

11.1. Normative References

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

[To22] Touch, J., W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo (work in progress), Oct. 2022

[RFC9293] Eddy, W. (Ed.), "Transmission Control Protocol", STD 7, RFC 9293, Aug. 2022.

11.2. Informative References

[Br14] Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

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

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and Paasch, C., "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, January 2013.

[Yo11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

12. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback.

The use of multiple segments to extend the option space of a SYN was initially proposed by Bob Briscoe. His initial proposal used complementary SYNs in an earlier version of this document, which evolved into mutually-exclusive "Sister-SYNs" in [Br14].

This work is partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Ted Faber
Engineering Specialist
Computer Systems Research Department
The Aerospace Corporation
2310 E. El Segundo Blvd.
El Segundo, CA 90245-4609 USA

Phone: +1 (310) 336-7373
Email: theodore.v.faber@aero.org