TCP Low Latency Option
draft-wang-tcpm-low-latency-opt-00

Abstract

This document specifies the TCP Low Latency option, which TCP connections can use during the connection establishment handshake to communicate extra parameters that can improve performance in low-latency environments. With the first such parameter, a TCP data receiver can advertise a hint about the Maximum ACK Delay (MAD) it will schedule for its own delayed ACK mechanism. This enables the TCP data sender to achieve lower latencies during loss recovery by using the Maximum ACK Delay advertised by the remote receiver to help compute retransmission timeouts that are potentially much lower than would otherwise be feasible. The Low Latency option is extensible, and later versions of this draft will introduce other mechanisms, including TCP timestamps with a finer granularity than those supported by RFC 7323.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 10, 2017.
1. Introduction

TCP receivers typically implement a delayed ACK algorithm, as specified in [RFC1122] Sec 4.2.3.2; as summarized in [RFC5681] sec 4.2, "an ACK SHOULD be generated for at least every second full-sized segment, and MUST be generated within 500 ms of the arrival of the first unacknowledged packet." In practice, many widely-deployed implementations have tended to delay ACKs by up to roughly 200ms. This is probably a historical artifact inherited from the 200ms "fast timeout" mechanism in the BSD TCP implementation from the late 1980s [WS95].

As a result, to avoid spurious timeouts due to delayed ACKs, widely-deployed TCP sender implementations have adapted to this delayed ACK behavior by constraining retransmission timeout (RTO) values to be at least 200ms.

Unfortunately, this 200ms value is 2000x the typical RTT of today’s commodity datacenter networks (which are typically below 100 microseconds). So senders constraining RTOs to be at least 200ms are paying a latency penalty much higher than the RTT in such environments.

The TCP Low Latency option enables a TCP data receiver to advertise a hint about the Maximum ACK Delay (MAD) it will schedule for its own delayed ACK mechanism. The receiver specifies the MAD value in the Low Latency option because the value that is feasible can be quite different for different receivers, based on the CPU’s speed, CPU and network workloads, and OS-specific constraints on minimum supported timer granularity.

This Low Latency option enables the TCP data sender to achieve lower latencies during loss recovery by using the Maximum ACK Delay.
advertised by the remote receiver to help compute retransmission timeouts that are potentially much lower than would otherwise be feasible.

2. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In this document, "MAD" refers to the Maximum Ack Delay used by the data receiver to delay TCP acknowledgments, and "minRTO" refers to the Minimum Retransmit Timeout.

3. Detailed Protocol

3.1. TCP Low Latency Option

The Low Latency option is only valid in SYN or SYN/ACK packets during the three way handshake. It MUST be ignored in other cases.

The format of the TCP Low Latency option is as follows:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Kind     |    Length     |M u|        MAD        |       |
|               |               |A n|       Value       |  Res  |
|               |               |D i|     (10 bits)     |       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        ...  Reserved  ...                     ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Kind: 1 byte: value = IANA-assigned option number
Length: 1 byte: value = 4 (or longer in later versions)
MAD unit: 2 bits: indicates time unit for MAD value:
0: reserved
1: milliseconds
2: microseconds
3: nanoseconds
MAD value: 10 bits: indicates MAD value set on the host:
1 ... 1023: MAD value in the given units
0: no MAD value is specified
Reserved: N>=4 bits: value = 0
In order to support future extensions, the option is variable-length. Bits beyond those defined so far in IETF standards should be considered "reserved". TCP implementations MUST (a) set to zero any reserved bits they add for padding, and (b) ignore any reserved bits (whether they are set or not).

3.2. Overview

The communication, starting from the TCP connection handshake, looks like the following:

TCP A (Active)                                  TCP B (Passive)
=============                                  ===============
CLOSED                                          LISTEN
#1 SYN-SENT       ----- <SYN,MAD=10ms>  ------> SYN-RCVD
(Adjust RTO accordingly)
#2 ESTABLISHED    <---- <SYN,ACK,MAD=5ms> ----- SYN-RCVD
(Adjust RTO accordingly)
#3 ESTABLISHED    -------<ACK>----------------> ESTABLISHED
#4 Send()         --------<DATA-1>------------> -
                      | Delay Ack < 5ms
                      |<--------<ACK-1>--------------- -
#5                  Recv()

#6 Send()          --------<DATA-2>------------>
                    | RTO >= 5ms
                    | |<---<DATA-2 retransmit>---->
                    |<--------<ACK-2>---------------
#7                  Recv()

3.3. Configuring maximum ACK delay

An implementation that supports the maximum ACK delay parameter MUST provide a user API to configure the maximum ACK delay for a specific connection or all TCP connections.

- If the user does not specify a MAD value, then the implementation SHOULD NOT specify a MAD value in the Low Latency option.

- If the user specifies a MAD value outside the range of ACK delay values supported by the implementation, then the implementation SHOULD allow the request to succeed, but SHOULD silently constrain the MAD value to be within the valid range (between the minimum and maximum ACK delay for the implementation). This is intended
to allow applications to portably request a MAD value without needing special logic to search for a valid value.

- If the specified connections are not in CLOSED or LISTEN states, the API SHOULD return an error and ignore the request to specify a MAD value.

- Otherwise the implementation SHOULD use the user-specified value as the maximum timeout for the delayed ACK and the MAD value in the Low Latency option of the specified TCP connections.

The exact design and implementation of such an API is intentionally left to the implementation. We discuss some examples in the appendix.

### 3.4. Announcing the maximum ACK delay

- The maximum ACK delay is announced to the remote TCP endpoint by including a Low Latency option with a non-zero MAD value in the SYN or SYN/ACK packet. A "MAD value" field of 0 in the Low Latency option indicates that the sender is not specifying a MAD value.

- If specified, then the MAD value in the Low Latency option MUST be set, as close as possible, to the implementation’s actual delayed ACK timeout for the connection. Note that the actual maximum delayed ACK timeout of the connection may be larger than the actual user specified value because of implementation constraints (e.g. timer granularity limitations).

- If the user has specified a MAD value for an active connection, then the active open side SHOULD include a Low Latency option with a MAD value in the SYN packet.

- If the user has specified a MAD value for a passive connection, and the passive side has received at least one SYN packet with a Low Latency option with a valid MAD value, then the passive open side SHOULD return its MAD value in the Low Latency option.

### 3.5. Adjusting TCP retransmission timeouts

If the MAD value advertised in a received Low Latency option is 0, or greater than the default maximum ACK delay of 200ms, then the option SHOULD be ignored and no further action is needed.

Otherwise the (data) sender MAY use the maximum delayed ACK advertised by the receiver to adjust the sender’s RTO calculation. Specifically, if the sender implements an RTO calculation based on
[RFC6298], it MAY replace the 1 second lower-bound specified in step 2.4 in Section 2 with the value of the maximum ACK delay advertised in the Low Latency option, so that the calculation becomes:

\[
RTO \leftarrow \text{SRTT} + \max(G, K \times \text{RTTVAR}) + \max(G, \text{max_ACK_delay})
\]

instead of

\[
RTO \leftarrow \max(\text{SRTT} + \max(G, K \times \text{RTTVAR}), 1 \text{ second}) /* [RFC6298] */
\]

Here we use the notation of [RFC6298], including SRTT (smoothed round-trip time), RTTVAR (round-trip time variation), and G (clock granularity).

Also, if the sender also implements [draft-ietf-tcpm-rack] then it SHOULD replace the maximum delayed ACK parameter (WCDelAckT) with the max_ACK_delay specified in the Low Latency option.

Using the MAD value in the RTO calculation helps senders reduce the RTO significantly while still avoiding spurious retransmissions due to delayed acks. With this new algorithm, the RTO can be drastically shortened in most environments where the receiver advertises a MAD. In particular, in data center environments the RTO can often be reduced from more than one second to single-digit milliseconds. Using the MAD to reduce the RTO can improve performance and thus mitigate TCP incast issues. More details are provided in the following Related work section.

4. Related work

Several research papers have shown that reducing the minimum retransmission timeout (minRTO) significantly improves the performance of TCP in the datacenter, by mitigating the effect of TCP timeouts. As a result, this can mitigate TCP incast issues.

- In "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter" [JS15], the authors show that reducing minRTO from 200ms to 5ms greatly reduced the impact of TCP incast issues.

- In "Understanding TCP incast throughput collapse in datacenter networks" [CG09], the authors show significant improvement in goodput when reducing minRTO.

- In "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems" [PK07], the authors show that reducing minRTO from 200 milliseconds to 200 microseconds improved goodput by an order of magnitude in some data center scenarios they evaluated.
In "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication" [VP09], the authors point out that the imbalance between the TCP minRTO and datacenter latencies can result in poor performance for applications sensitive to millisecond-scale delays in query response times. In simulations of datacenter scenarios they show that goodput drops when increasing minRTO above 1ms. Moreover, in some data center scenarios the default minRTO of 200ms results in nearly 2 orders of magnitude lower throughput compared to a minRTO of 1ms.

In Google data centers a TCP option mechanism equivalent to the Low Latency option's MAD parameter has been used since 2005, and the TCP minRTO has been set to 5ms by default since 2013 [CC16].

5. Middlebox Considerations

The new Low Latency option might expose some middlebox issues:

- Middleboxes could drop SYN s with a Low Latency option in the case where it treats the Low Latency option as an unknown option. However, this happens fairly rarely according to "Is it still possible to extend TCP?" [HN11], table 3.

- In case middleboxes alter the content in the Low Latency option, the receiver SHOULD do a sanity check on the MAD value included in the Low Latency option to verify it is less than or equal to the default maximum ACK delay of 200ms. As explained earlier, it is not practical for users to set MAD value greater than default. So it is safe to consider a MAD value greater than default as a result of a bad user configuration or a malfunctioning middlebox and ignore the Low Latency option completely in such cases.

6. Security Considerations

TBD

7. IANA Considerations

As no official option number has been issued for the new Low Latency option by IANA yet, experimental option 254 per [RFC6994] with magic number 0xF990 (16 bits) is used for now.

The option format with experimental ID is as follows:
8. Appendix

8.1. Example user API in Linux to configure maximum ACK delay

8.1.1. Per-route MAD configuration API

A new configuration option called "mad" will be added to the "ip" command line tool in the iproute2 package. Users can use this to configure a per-route MAD value like the following:

ip route add 10.1.2.0/24 dev eth0 scope link src 10.1.2.123 mad 5ms

This configures all connections destined to 10.1.2.0/24 to have a MAD value of 5ms. When implementing this new MAD option field, the "ip" command line tool will verify that the provided MAD parameter is less than or equal to the default MAD value of 200ms. If the MAD is invalid then the ip route command will ignore the command and report an error to user.

Newly-created TCP sockets have the default 200ms MAD value. When a TCP connection is opened, it SHOULD consult the ip routing table to check if there is any configured MAD value for the route. If so, the
implementation copies the route’s MAD value to the connection’s MAD value.

This per-route configuration will mostly be used by network administrators when configuring routes on the host.

8.1.2. MAD Socket option API

Socket options provide per-connection configuration parameters. To allow per-connection configuration of the MAD value in the Low Latency option, a new TCP socket option called TCP_MAD will be added to the TCP implementation. This will allow applications to request a MAD value on a finer granularity than the per-route configuration, depending on the application’s requirements.

The API will look like the following example:

```c
int mad_val = 5 * 1000 * 1000; // in ns unit: 5ms
err = setsockopt(fd, SOL_TCP, TCP_MAD, &mad_val, sizeof(mad_val));
```

The socket option implementation will sanitize the MAD value provided by the user. Per the specification above, in the "Configuring maximum ACK delay" section, if the user specifies a MAD value outside the range of ACK delay values supported by the implementation, then the implementation will allow the request to succeed, but will silently constrain the MAD value to be within the valid range (between the minimum and maximum ACK delay for the implementation). This is intended to allow applications to portably request a MAD value without needing special logic to search for a valid value.

Once the implementation has sanitized the provided MAD value, it will record the value in the socket as the socket’s own MAD value.

Note: the MAD value set by the socket option SHOULD always override the per-route MAD value if there is one.

9. References

9.1. Normative References

[draft-ietf-tcpm-rack]


9.2. Informative References


Authors’ Addresses

Wei Wang
Google, Inc
1600 Amphitheater Parkway
Mountain View, California 94043
USA

Email: weiwan@google.com