Virtual reassembly buffers in 6LoWPAN

draft-bormann-lwig-6lowpan-virtual-reassembly-00

Abstract

When employing adaptation layer fragmentation in 6LoWPAN, it may be beneficial for a forwarder not to have to reassemble each packet in its entirety before forwarding it.

This has been always possible with the original fragmentation design of RFC 4944. Apart from a brief mention of the way to do this in Section 2.5.2 of the 6LoWPAN book, this has not been extensively described in the literature. The present document attempts to fill that gap.

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 July 29, 2018.

Copyright Notice

Copyright (c) 2018 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
1. Introduction

(TO DO: Insert an extended form of the abstract first here, expanding the references to [RFC4944] and [BOOK] in the process.)

2. Reassembly buffers

An adaptation layer implementation for 6LoWPAN needs to perform reassembly of every fragmented packet received in order to be able to forward the packet (re-fragmenting it in the process).

A reassembly buffer for 6LoWPAN contains:

- `datagram_size`,
- `datagram_tag` and L2 sender and receiver addresses (to which the `datagram_tag` is local),
- actual packet data from the fragments received so far, in a form that makes it possible to detect when the whole packet has been received and can be processed or forwarded,
- a timer that allows discarding the partial packet after a timeout.

This requires a reassembly buffer for each fragmented packet the reception of which is in progress. Since the forwarder may be receiving fragments for multiple packets concurrently (e.g., from different senders), this means that multiple reassembly buffers are needed, easily dominating the memory requirements in a 6LoWPAN.
implementation. Worse, as this space may still be limited, any lack of reassembly buffers may lead to an increased loss rate for fragmented packets (which already have to cope with a higher compound loss rate).

3. Virtual reassembly

To reduce the memory requirement for reassembly buffers, the implementation may opt to not keep the actual packet data in the reassembly buffer. Instead, it may attempt to send out the data for a fragment in the form of a forwarded fragment, as soon as all necessary information for that is available. Obviously, all fragments need to be sent with the same outgoing address (otherwise a full reassembly implementation would discard the fragments) and the same datagram_tag.

To this end, the reassembly buffer now also stores, as soon as enough of the packet is available to make a forwarding decision (i.e., as soon as the first fragment has been received):

- L2 destination address used for forwarding,
- outgoing datagram_tag chosen for this packet.

A simple implementation may do away with any attempt to keep packet data in the virtual reassembly buffer. It then has to discard all non-first fragments for which a reassembly buffer is not already available (penalizing reordering, which however may be rare).

Note that the decision to do local processing of a packet needs to be taken with the first fragment — such packets of course do need to be fully reassembled (unless transport and application also can cope with fragments, which they rarely can in the presence of security).

4. Header compression

[RFC6282] defines the header compression format for 6LoWPAN. One important impact of header compression is that the header is no longer of a fixed length. In particular, changes made by a forwarder may gain or lose the ability to use a more highly compressed variant, changing the length of the header in the packet. If the change increases the size, the maximum frame size may be exceeded, leading to the need to re-fragment in the forwarder. This is less of a problem with full reassembly, but with virtual reassembly can lead to the need for sending an additional frame for each packet.

The well-known approach to minimize the probability of this need is for the original sender to put all slack in the frame sizes into the
Internet-Draft    Virtual reassembly buffers in 6LoWPAN     January 2018

_first_ packet, making this the smallest fragment and not the last
one as would be done in a naive implementation. (This also has other
consequences related to delivery probability, which are not discussed
here.) This makes sure an additional fragment only needs to be sent
if the header expansion during forwarding would have created an
additional fragment with full reassembly as well.

5. IANA Considerations

This document makes no requests of IANA.

6. Security considerations

TBD

7. References

7.1. Normative References

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
"Transmission of IPv6 Packets over IEEE 802.15.4
Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,

7.2. Informative References

[BOOK]     Shelby, Z. and C. Bormann, "6LoWPAN", John Wiley & Sons,

Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
DOI 10.17487/RFC6282, September 2011,

Acknowledgements

Many people have mentioned that it would be good to have a
description of virtual reassembly in 6LoWPAN. Finally, Thomas
Watteyne assembled a design team that intends to work on 6Lo
fragmentation. Writing up the present document has been motivated by
that work.

Author’s Address
TCP over Constrained-Node Networks
draft-gomez-lwig-tcp-constrained-node-networks-03

Abstract

This document provides a profile for the Transmission Control Protocol (TCP) over Constrained-Node Networks (CNNs). The overarching goal is to offer simple measures to allow for lightweight TCP implementation and suitable operation in such environments.

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 31, 2017.

Copyright Notice

Copyright (c) 2017 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

Copyright (c) 2017 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
1. Introduction

The Internet Protocol suite is being used for connecting Constrained-Node Networks (CNNs) to the Internet, enabling the so-called Internet of Things (IoT) [RFC7228]. In order to meet the requirements that stem from CNNs, the IETF has produced a suite of protocols specifically designed for such environments [I-D.ietf-lwig-energy-efficient].

At the application layer, the Constrained Application Protocol (CoAP) was developed over UDP [RFC7252]. However, the integration of some CoAP deployments with existing infrastructure is being challenged by middleboxes such as firewalls, which may limit and even block UDP-
based communications. This is the main reason why a CoAP over TCP specification is being developed [I-D.tschofenig-core-coap-tcp-tls].

On the other hand, other application layer protocols not specifically designed for CNNs are also being considered for the IoT space. Some examples include HTTP/2 and even HTTP/1.1, both of which run over TCP by default [RFC7540][RFC2616], and the Extensible Messaging and Presence Protocol (XMPP) [RFC 6120]. TCP is also used by non-IETF application-layer protocols in the IoT space such as MQTT and its lightweight variants [MQTTs].

This document provides a profile for TCP over CNNs. The overarching goal is to offer simple measures to allow for lightweight TCP implementation and suitable operation in such environments.

1.1. 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 [RFC2119]

2. Characteristics of CNNs relevant for TCP

CNNs are defined in [RFC7228] as networks whose characteristics are influenced by being composed of a significant portion of constrained nodes. The latter are characterized by significant limitations on processing, memory, and energy resources, among others [RFC7228]. The first two dimensions pose constraints on the complexity and on the memory footprint of the protocols that constrained nodes can support. The latter requires techniques to save energy, such as radio duty-cycling in wireless devices [I-D.ietf-lwig-energy-efficient], as well as minimization of the number of messages transmitted/received (and their size).

Constrained nodes often use physical/link layer technologies that have been characterized as ‘lossy’. Many such technologies are wireless, therefore exhibiting a relatively high bit error rate. However, some wired technologies used in the CNN space are also lossy (e.g. Power Line Communication). Transmission rates of CNN radio or wired interfaces are typically low (e.g. below 1 Mbps).

Some CNNs follow the star topology, whereby one or several hosts are linked to a central device that acts as a router connecting the CNN to the Internet. CNNs may also follow the multihop topology [RFC6606].
3. Scenario

The main scenario for use of TCP over CNNs comprises a constrained device and an unconstrained device that communicate over the Internet using TCP, possibly traversing a middlebox (e.g. a firewall, NAT, etc.). Figure 1 illustrates such scenario. Note that the scenario is asymmetric, as the unconstrained device will typically not suffer the severe constraints of the constrained device. The unconstrained device is expected to be mains-powered, to have high amount of memory and processing power, and to be connected to a resource-rich network.

Assuming that a majority of constrained devices will correspond to sensor nodes, the amount of data traffic sent by constrained devices (e.g. sensor node measurements) is expected to be higher than the amount of data traffic in the opposite direction. Nevertheless, constrained devices may receive requests (to which they may respond), commands (for configuration purposes and for constrained devices including actuators) and relatively infrequent firmware/software updates.

Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

4. TCP over CNNs

4.1. TCP connection initiation

In the constrained device to unconstrained device scenario illustrated above, a TCP connection is typically initiated by the constrained device, in order for this device to support possible sleep periods to save energy.
4.2. Maximum Segment Size (MSS)

Some link layer technologies in the CNN space are characterized by a short data unit payload size, e.g. up to a few tens or hundreds of bytes. For example, the maximum frame size in IEEE 802.15.4 is 127 bytes.

6LoWPAN defined an adaptation layer to support IPv6 over IEEE 802.15.4 networks. The adaptation layer includes a fragmentation mechanism, since IPv6 requires the layer below to support an MTU of 1280 bytes [RFC2460], while IEEE 802.15.4 lacked fragmentation mechanisms. 6LoWPAN defines an IEEE 802.15.4 link MTU of 1280 bytes [RFC4944]. Other technologies, such as Bluetooth LE [RFC7668], ITU-T G.9959 [RFC7428] or DECT-ULE [RFC8105], also use 6LoWPAN-based adaptation layers in order to enable IPv6 support. These technologies do support link layer fragmentation. By exploiting this functionality, the adaptation layers that enable IPv6 over such technologies also define an MTU of 1280 bytes.

For devices using technologies with a link MTU of 1280 bytes (e.g. defined by a 6LoWPAN-based adaptation layer), in order to avoid IP layer fragmentation, the TCP MSS must not be set to a value greater than 1220 bytes in CNNs, and it must not be set to a value leading to an IPv6 datagram size exceeding 1280 bytes. (Note: IP version 6 is assumed.)

On the other hand, there exist technologies also used in the CNN space, such as Master Slave / Token Passing (TP) [RFC8163], Narrowband IoT (NB-IoT) [I-D.ietf-lpwan-overview] or IEEE 802.11ah [I-D.delcarpio-6lo-wlanah], that do not suffer the same degree of frame size limitations as the technologies mentioned above. The MTU for MS/TP is recommended to be 1500 bytes [RFC8163], the MTU in NB-IoT is 1600 bytes, and the maximum frame payload size for IEEE 802.11ah is 7991 bytes. Over such technologies, the TCP MSS may be set to a value greater than 1220 bytes, as long as IPv6 datagram size does not exceed the MTU for each technology. One consideration in this regard is that, when a node supports an MTU greater than 1280 bytes, it 'SHOULD' then support Path MTU (PMTU) discovery [RFC1981]. (Note that, as explained in RFC 1981, a minimal IPv6 implementation may 'choose to omit implementation of Path MTU Discovery'.) For the sake of lightweight implementation and operation, unless applications require handling large data units (i.e. leading to an IPv6 datagram size greater than 1280 bytes), it may be desirable to limit the MTU to 1280 bytes.
4.3. Window Size

A TCP stack can reduce the implementation complexity by advertising a TCP window size of one MSS, and also transmit at most one MSS of unacknowledged data, at the cost of decreased performance. This size for receive and send window is appropriate for simple message exchanges in the CNN space, reduces implementation complexity and memory requirements, and reduces overhead (see section 4.7).

A TCP window size of one MSS follows the same rationale as the default setting for NSTART in [RFC7252], leading to equivalent operation when CoAP is used over TCP.

For devices that can afford greater TCP window size, it may be useful to allow window sizes of at least five MSSs, in order to allow Fast Retransmit and Fast Recovery [RFC5681].

4.4. RTO estimation

If a TCP sender uses very small window size and cannot use Fast Retransmit/Fast Recovery or SACK, the RTO algorithm has a larger impact on performance than for a more powerful TCP stack. In that case, RTO algorithm tuning may be considered, although careful assessment of possible drawbacks is recommended. A fundamental trade-off exists between responsiveness and correctness of RTOs [I-D.ietf-tcpm-rto-consider]. A more aggressive RTO behavior reduces wait time before retransmissions, but it also increases the probability of incurring spurious timeouts. The latter lead to unnecessary waste of potentially scarce resources in CNNs such as energy and bandwidth.

On a related note, there has been recent activity in the area of defining an adaptive RTO algorithm for CoAP (over UDP). As shown in experimental studies, the RTO estimator for CoAP defined in [I-D.ietf-core-cocoa] (hereinafter, CoCoA RTO) outperforms state-of-art algorithms designed as improvements to RFC 6298 [RFC6298] for TCP, in terms of packet delivery ratio, settling time after a burst of messages, and fairness (the latter is specially relevant in multihop networks connected to the Internet through a single device, such as a 6LoWPAN Border Router (6LBR) configured as a RPL root) [Commag]. In fact, CoCoA RTO has been designed specifically considering the challenges of CNNs, in contrast with the RFC 6298 RTO.
4.5. TCP connection lifetime

[[Note: future revisions will better separate what a TCP stack should support, or not, and how the TCP stack should be used by applications, e.g., whether to close connections or not.]]

4.5.1. Long TCP connection lifetime

In CNNs, in order to minimize message overhead, a TCP connection should be kept open as long as the two TCP endpoints have more data to exchange or it is envisaged that further segment exchanges will take place within an interval of two hours since the last segment has been sent. A greater interval may be used in scenarios where applications exchange data infrequently.

TCP keep-alive messages [RFC1122] may be supported by a server, to check whether a TCP connection is active, in order to release state of inactive connections. This may be useful for servers running on memory-constrained devices.

Since the keep-alive timer may not be set to a value lower than two hours [RFC1122], TCP keep-alive messages are not useful to guarantee that filter state records in middleboxes such as firewalls will not be deleted after an inactivity interval typically in the order of a few minutes [RFC6092]. In scenarios where such middleboxes are present, alternative measures to avoid early deletion of filter state records (which might lead to frequent establishment of new TCP connections between the two involved endpoints) include increasing the initial value for the filter state inactivity timers (if possible), and using application layer heartbeat messages.

4.5.2. Short TCP connection lifetime

A different approach to addressing the problem of traversing middleboxes that perform early filter state record deletion relies on using TCP Fast Open (TFO) [RFC7413]. In this case, instead of trying to maintain a TCP connection for long time, possibly short-lived connections can be opened between two endpoints while incurring low overhead. In fact, TFO allows data to be carried in SYN (and SYN-ACK) packets, and to be consumed immediately by the receiving endpoint, thus reducing overhead compared with the traditional three-way handshake required to establish a TCP connection.

For security reasons, TFO requires the TCP endpoint that will open the TCP connection (which in CNNs will typically be the constrained device) to request a cookie from the other endpoint. The cookie, with a size of 4 or 16 bytes, is then included in SYN packets of subsequent connections. The cookie needs to be refreshed (and
obtained by the client) after a certain amount of time. Nevertheless, TFO is more efficient than frequently opening new TCP connections (by using the traditional three-way handshake) for transmitting new data, as long as the cookie update rate is well below the data new connection rate.

4.6. Explicit congestion notification

Explicit Congestion Notification (ECN) [RFC3168] may be used in CNNs. ECN allows a router to signal in the IP header of a packet that congestion is arising, for example when queue size reaches a certain threshold. If such a packet encapsulates a TCP data packet, an ECN-enabled TCP receiver will echo back the congestion signal to the TCP sender by setting a flag in its next TCP ACK. The sender triggers congestion control measures as if a packet loss had happened. In that case, when the congestion window of a TCP sender has a size of one segment, the TCP sender resets the retransmit timer, and will only be able to send a new packet when the retransmit timer expires [RFC3168]. Effectively, the TCP sender reduces at that moment its sending rate from 1 segment per Round Trip Time (RTT) to 1 segment per default RTO.

ECN can reduce packet losses, since congestion control measures can be applied earlier than after the reception of three duplicate ACKs (if the TCP sender window is large enough) or upon TCP sender RTO expiration [RFC2884]. Therefore, the number of retries decreases, which is particularly beneficial in CNNs, where energy and bandwidth resources are typically limited. Furthermore, latency and jitter are also reduced.

ECN is particularly appropriate in CNNs, since in these environments transactional type interactions are a dominant traffic pattern. As transactional data size decreases, the probability of detecting congestion by the presence of three duplicate ACKs decreases. In contrast, ECN can still activate congestion control measures without requiring three duplicate ACKs.

4.7. TCP options

A TCP implementation needs to support options 0, 1 and 2 [RFC793]. A TCP implementation for a constrained device that uses a single-MSS TCP receive or transmit window size may not benefit from supporting the following TCP options: Window scale [RFC1323], TCP Timestamps [RFC1323], Selective Acknowledgements (SACK) and SACK-Permitted [RFC2018]. Other TCP options should not be used, in keeping with the principle of lightweight operation.
Other TCP options should not be supported by a constrained device, in keeping with the principle of lightweight implementation and operation.

If a device, with less severe memory and processing constraints, can afford advertising a TCP window size of several MSSs, it may support the SACK option to improve performance. SACK allows a data receiver to inform the data sender of non-contiguous data blocks received, thus a sender (having previously sent the SACK-Permitted option) can avoid performing unnecessary retransmissions, saving energy and bandwidth, as well as reducing latency. The receiver supporting SACK will need to manage the reception of possible out-of-order received segments, requiring sufficient buffer space.

SACK adds 8*n+2 bytes to the TCP header, where n denotes the number of data blocks received, up to 4 blocks. For a low number of out-of-order segments, the header overhead penalty of SACK is compensated by avoiding unnecessary retransmissions.

Another potentially relevant TCP option in the context of CNNs is (TFO) [RFC7413]. As described in section 4.5.2, TFO can be used to address the problem of traversing middleboxes that perform early filter state record deletion.

4.8. Delayed Acknowledgments

A device that advertises a single-MSS receive window needs to avoid use of delayed ACKs in order to avoid contributing unnecessary delay (of up to 500 ms) to the RTT [RFC5681].

When traffic over a CNN is expected to be mostly of transactional type, with transaction size typically below one MSS, delayed ACKs are not recommended. For transactional-type traffic between a constrained device and a peer (e.g. backend infrastructure) that uses delayed ACKs, the maximum ACK rate of the peer will be typically of one ACK every 200 ms (or even lower). If in such conditions the peer device is administered by the same entity managing the constrained device, it is recommended to disable delayed ACKs at the peer side.

On the other hand, delayed ACKs allow to reduce the number of ACKs in bulk transfer type of traffic, e.g. for firmware/software updates or for transferring larger data units containing a batch of sensor readings.
4.9. Explicit loss notifications

There has been a significant body of research on solutions capable of explicitly indicating whether a TCP segment loss is due to corruption, in order to avoid activation of congestion control mechanisms [ETEN] [RFC2757]. While such solutions may provide significant improvement, they have not been widely deployed and remain as experimental work. In fact, as of today, the IETF has not standardized any such solution.

5. Security Considerations

If TFO is used, the security considerations of RFC 7413 apply.

There exist TCP options which improve TCP security. Examples include the TCP MD5 signature option [RFC2385] and the TCP Authentication Option (TCP-AO) [RFC5925]. However, both options add overhead and complexity. The TCP MD5 signature option adds 18 bytes to every segment of a connection. TCP-AO typically has a size of 16-20 bytes.

6. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

The authors appreciate the feedback received for this document. The following folks provided comments that helped improve the document: Carsten Bormann, Zhen Cao, Wei Genyu, Michael Scharf, Ari Keranen, Abhijan Bhattacharyya, Andres Arcia-Moret, Yoshifumi Nishida, Joe Touch, Fred Baker, Nik Sultana, Kerry Lynn, and Erik Nordmark. Simon Brummer provided details on the RIOT TCP implementation. Xavi Vilajosana provided details on the OpenWSN TCP implementation.

7. Annex. TCP implementations for constrained devices

This section overviews the main features of TCP implementations for constrained devices.

7.1. uIP

uIP is a TCP/IP stack, targetted for 8 and 16-bit microcontrollers. uIP has been deployed with Contiki and the Arduino Ethernet shield.
A code size of ˜5 kB (which comprises checksumming, IP, ICMP and TCP) has been reported for uIP [Dunk].

uIP provides a global buffer for incoming packets, of single-packet size. A buffer for outgoing data is not provided. In case of a retransmission, an application must be able to reproduce the same packet that had been transmitted.

The MSS is announced via the MSS option on connection establishment and the receive window size (of one MSS) is not modified during a connection. Stop-and-wait operation is used for sending data. Among other optimizations, this allows to avoid sliding window operations, which use 32-bit arithmetic extensively and are expensive on 8-bit CPUs.

7.2. lwIP

lwIP is a TCP/IP stack, targeted for 8- and 16-bit microcontrollers. lwIP has a total code size of ˜14 kB to ˜22 kB (which comprises memory management, checksumming, network interfaces, IP, ICMP and TCP), and a TCP code size of ˜9 kB to ˜14 kB [Dunk].

In contrast with uIP, lwIP decouples applications from the network stack. lwIP supports a TCP transmission window greater than a single segment, as well as buffering of incoming and outgoing data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit and fast recovery. SACK and Window Scale have been recently added to lwIP.

7.3. RIOT

The RIOT TCP implementation (called GNRC TCP) has been designed for Class 1 devices [RFC 7228]. The main target platforms are 8- and 16-bit microcontrollers. GNRC TCP offers a similar function set as uIP, but it provides and maintains an independent receive buffer for each connection. In contrast to uIP, retransmission is also handled by GNRC TCP. GNRC TCP uses a single-MSS window size, which simplifies the implementation. The application programmer does not need to know anything about the TCP internals, therefore GNRC TCP can be seen as a user-friendly uIP TCP implementation.

The MSS is set on connections establishment and cannot be changed during connection lifetime. GNRC TCP allows multiple connections in parallel, but each TCB must be allocated somewhere in the system. By default there is only enough memory allocated for a single TCP connection, but it can be increased at compile time if the user needs multiple parallel connections.
7.4. OpenWSN

The TCP implementation in OpenWSN is mostly equivalent to the uIP TCP implementation. OpenWSN TCP implementation only supports the minimum state machine functionality required. For example, it does not perform retransmissions.

7.5. TinyOS

TBD

7.6. Summary

<table>
<thead>
<tr>
<th>OS</th>
<th>uIP</th>
<th>lwIP orig</th>
<th>lwIP 2.0</th>
<th>RIOT</th>
<th>OpenWSN</th>
<th>Tiny</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Data size</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Code size (kB)</td>
<td>&lt; 5</td>
<td>~9 to ~14</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Window size (MSS)</td>
<td>1</td>
<td>Multiple</td>
<td>Multiple</td>
<td>1</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>Slow start</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>Fast rec/retx</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>Keep-alive</td>
<td>No</td>
<td>*</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>TFO</td>
<td>No</td>
<td>No</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>ECN</td>
<td>No</td>
<td>No</td>
<td>*</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>Window Scale</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>TCP timestamps</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td>SACK</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>Delayed ACKs</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
</tr>
</tbody>
</table>

Figure 2: Summary of TCP features for different lightweight TCP implementations.
8. References

8.1. Normative References


8.2. Informative References


Carles Gomez
UPC/i2CAT
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu
Neighbor Management Policy for 6LoWPAN
draft-ietf-lwig-nbr-mgmt-policy-01

Abstract

This document describes the problems associated with neighbor cache management in constrained multihop networks and a sample neighbor management policy to deal with it.

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 August 26, 2018.

Copyright Notice

Copyright (c) 2018 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
1. Introduction

In a wireless multihop network, the node densities (maximum number of devices connected on a single hop) may vary significantly depending upon deployments/scenarios. While there is some policy control possible with regards to the network size in terms of maximum number of devices connected, it is especially difficult to set a figure on what will be the maximum node density given a deployment. For e.g. A network can put an upper limit on max 1000 devices but it is impossible to state what the node density will be in this 1000 node network.

A neighbor cache is used for populating neighboring one-hop connected nodes information such as MAC address, link local IP address and other reachability state information. Node density has direct implications on the neighbor cache and in constrained network scenario the size of the neighbor cache will be limited. Thus there are chances that a node may not be able to fit all the neighboring nodes in the neighbor cache.

Jadhav, et al.
nodes in its cache in which case it has to prioritize entries and thus needs a neighbor management policy.

This draft presents problems related to neighbor management policies by considering a security-enabled multi-hop 6lo network. This document considers RPL [RFC6550] as a routing protocol and PANA (EAP-PANA) [RFC5191] as a network access protocol. For RPL, both the storing and non-storing mode of operations are considered. We also provide a sample neighbor management policy which can be used in such networks and its limitations. The aim of such a policy is to retain set of neighbor cache entries with high quality links such that routing adjacencies are stabilized.

The problem statement and the proposed solution described is also relevant to other multihop constrained scenarios such as 6TiSCH [I-D.ietf-6tisch-architecture]. [I-D.ietf-6tisch-minimal-security] talks about a pledge (new joinee) node authenticating via a Join Proxy. The considerations mentioned in this document are applicable for such networks as well.
1.1. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

PaC (PANA Client): New joining node which is yet to be authenticated.

PRE (PANA Relay Element): An already authenticated and network joined node which is willing to act as a relay element for PaCs to complete their authentication procedure on multi-hop networks. [RFC6345] describes the details of PRE.

PAA (PANA Auth Agent): Auth server which hosts the credentials database. PaC will handshake with PAA to complete authentication procedure.
Routing Child: A downstream node who is part of the routing table of the parent. For e.g. in the sample topology above N5 is the directly connected routing child for N3. N6 and N7 are also part of N3 routing table, they are routing child nodes but not directly connected. For N6 and N7 the document might alternatively use a term grand-child.

Routing Parent: In Figure 1, N1 and N2 are possible routing parents for N3.

Neighbor Cache Entry (NCE): A neighbor entry managed on behalf of directly connected peer.

This document also uses terminology described in [RFC6550] and [RFC6775].

2. Neighbor Management

2.1. Significance of Neighbor management policy

Multihop mesh networks present unique challenges to neighbor management especially with resource constrained nodes. In cases where the node density is higher than the neighbor cache size, the entries have to be prioritized. [Woo_et_al] and [Dawans_et_al] talk about prioritization of neighbor entries by using link quality estimation techniques. But prioritization alone may not necessarily be optimal in all cases. The reason or function why neighbor entry was added also needs to be taken in consideration. For example, evicting a routing direct child might have a ripple effect in turn impacting all the sub-childs as well.

In case of key management protocols deployed above MAC layer in multihop network, the neighbor management kicks in early even before the routing adjacencies are established. Since a new joining node needs to discover/attach to a relay element for completing its authentication procedure, the neighbor cache entries have to be appropriately populated both on a PaC and on the PRE. If a neighbor entry whose authentication is in progress is evicted, it will negatively impact the authentication procedure.

Another important consideration is that with increased node density, the prioritization based on link estimation parameters might not help since there might be more well connected peers. In dense deployments the number of directly attached neighbors with good quality links might still be higher than the max entries in neighbor cache size.
2.2. Trivial neighbor management policies

This section investigates policies which are used by most of the current operating systems for constrained nodes. While such policies are trivial to implement they may not be able to deal with the constrained network scenario. Note that such policies can still be used if it is known apriori that the neighbor cache can hold entries for maximum node density.

a. First Come First Serve (FCFS) policy

b. Least Recently Used (LRU) policy

The primary distinction between these policies is how it treats a new entry when the neighbor cache is full. In case of FCFS policy, the new entry is simply rejected while with LRU, the new entry replaces the least recently used entry.

RPL works by initiating a downstream multicast DIO to establish upstream network path. Subsequently DAO messages might be sent by the nodes to establish downstream paths to the nodes. Thus the network is flooded with multicast DIO messages initially and similarly there are chances that the same node is ended up been selected as a preferred parent by most of the child nodes and thus receives a DAO message from all these child nodes. Note that once a node establishes a parent entry or a routing entry on behalf of a directly connected node then it has to also provision a neighbor cache entry for it for subsequent unicast traffic.

In case of FCFS policy, a node might end up hosting all the neighbor entries based on DIO or DAO messages. Once the cache is full all the subsequent attempts to add an NCE will fail.

In case of LRU policy, a node might end up churning lot of neighbor entries because once the cache gets full and there is a request for new entry, it would result in evicting the least recently used (but active) entry. If at later point of time, there is a traffic for the evicted entry then the old entry has to be reinstated using IPv6 NDP procedure. This would mean reinstating the entry by evicting another least recently used entry. If the node density is very high, then this churn would be substantially high to extent that it would disrupt any routing adjacencies to be established in the network in a stable way.
2.3. Lifecycle of a NCE

2.3.1. NCE Insertion

IPv6 NDP [RFC6775] defines signaling involved in resolving the IPv6 addresses to its corresponding MAC addresses which gets populated in the neighbor cache. In case of constrained network, it is desired that such control traffic is minimized and thus the neighbor cache entries are populated as part of existing messaging. One example would be when the node receives a DAO message from its immediate child node, it not only makes an addition to the routing table but also creates a neighbor cache entry for the node. Thus it eliminates need for additional IPv6 NDP NS/NA messaging involved to resolve MAC address. Similar heuristic is used to add neighbor entries in other cases as well. Section 10.3.2 of [RFC6775] describes update and addition of such NCEs based on routing information packets.

Following are the possible signaling scenarios in which case a neighbor entry may get added.

Node Joining procedure: A new joinee node discovers a relay element to initiate its auth procedure. At the end of the discovery phase the new joinee node would have known the link local IP address of the relay element. The joinee node will send an unsecured-NS to the relay element to solicit its NA. The PRE may send a NA with the suitable status code as defined in section 6.5.3 of [RFC6775].
Relay element does not hold any state information on behalf of the new joiner node except for its neighbor cache entry. Thus in the Figure 1 the new joinee node may select node N3 as its PRE, in which case N3 has to add a neighbor entry on behalf of the new joinee node.

Post authentication the node enters into network discovery phase. The node selects one or more of its neighboring peer as its preferred parent based on the DIO received from these peers. Note that the node’s selected relay element and its preferred parent may not be same. The preferred parent serves as a default router node to which all its upstream traffic is directed. Thus an NCE on behalf of preferred parent needs to be added. In Figure 1 node N5 selects N3 as its preferred parent. N5 needs to add neighbor entry on behalf of N3 which is its directly connected RPL preferred parent.

In case of RPL storing MOP (mode of operation), the node may send a DAO message containing its reachability information to its preferred parent. The parent node in turn may pass this information upstream to its parent by generating a DAO retaining the child node’s reachability information, establishing a downstream routing path towards the node who originated the DAO. The preferred parent has to maintain a neighbor entry on behalf of the directly connected child.
node. For example, in the Figure 1, node N3 needs to maintain a neighbor entry on behalf of N5 which is its directly connected child node. Nodes N6 and N7 are grand-child nodes for node N3 for whom no neighbor entry is required.

As mentioned in Section 10.3.2 of [RFC6775], the NCEs on parent and child can be added directly as a result of RPL DIO/DAO signalling without any explicit NS/NA messaging.

```
New  |  PRE  |  RPL  |  Parent  |  PAA
     |       |       |          |       
     |       |       |          |       |
     |       | AuthProc|          |       |
     |       |       |          |       |
     |       |       |          |       |
     |       | RPL-DIO |          |       |
     |       |       |          |       |
     |       |       |          |       |
     |       | addNCE(parent, reason=PARENT) |          |
     |       |       |          |       |
     |       | RPL-DAO |          |       |
     |       |       |          |       |
     |       |       |          |       |
     |       |       |          |       |
     |       |       |          |       |
     |       | addNCE(new, reason=CHILD) |          |
```

Figure 3: NCE creation call Flow for RPL storing MOP

In case of non-storing MOP, the parent node needs to know the global IPv6 address of the immediate child nodes. This is needed since the source routing header carries the global addresses and thus the NCE of the child node should contain the global address. Secondly, the RPL DAO is addressed directly to the root node in case of non-storing mode. Thus RPL messaging cannot be used for creating NCE entries on parent and child, unlike storing MOP. The child node may send a secure unicast NS with ARO option containing its global address to be registered on the parent node. The child node can still use RPL DIO to create an NCE on behalf of the parent node.
This document expects the neighbor management policy to remember the reason why the neighbor entry is inserted. Secondly, the router may remember whether the NS received was secured or unsecured and accordingly use it to prioritize eviction entries. As described in the next sections, this reason will help the policy to prioritize the entries in case an eviction is required.

### 2.3.2. NCE Deletion

It is imperative that an unwanted neighbor entry be removed as soon as possible. This section talks about different cases in which neighbor entry can be deleted.

Route Invalidation: In case of storing MOP, when the child node decides to switch its preferred parent, the RPL specifications allows the node to send a no-path DAO message to invalidate the route along the previous path(s). A directly connected parent node can use this message to clear the NCE. While the entry can be immediately cleared, usually the implementations choose to wait a small amount of time before clearing the entry. This is to avoid any impact on the in-transit traffic. Thus this also establishes the importance of route invalidation to achieve optimized neighbor cache utilization.
In case of non-storing mode, the no-path DAO cannot be not employed since the previous parent does not having any routing information to be invalidated. But the previous parent may still contain the NCE on behalf of the child node. This document recommends use of [RFC6775] section 6.5.3. which allows sending a zero lifetime ARO option in NS for deregistering the corresponding neighbor entry.

[RFC6775], ND optimizations for 6LoWPANs, section 5.5.3. talks about deleting the entries in case the NUD (neighbor unreachability detection) fails either due to no response to NS messages or due to failure response. NCEs in such cases should be deleted. An example where NUD NS would fail because of no response is the case where the child node switches its parent due to link unavailability. The parent in such a case would not receive the no-path DAO message or any other traffic from the child node. Thus on NCE lifetime expiry, the parent node would send NS which would fail with no response, thus triggering entry deletion.

2.3.3. NCE Eviction

The eviction rules have a major impact on the neighbor management policy. Eviction rules are used when the policy has to forcibly remove an active neighbor entry from the cache to make space for the new (hopefully higher priority) entry. The eviction policy may take into account several considerations such as the reason why the entry was made, is the entry in active use currently, how good (for e.g., based on link estimation) the entry currently is.

2.3.3.1. Eviction for directly connected routing entries

This section talks about implications of an eviction in which a parent node decides of evicting a directly connected routing child NCE. In the sample topology Figure 1, lets assume N3 needs to evict N5 from its neighbor cache. In case of RPL’s storing MOP, eviction of directly connected routing child NCE also has impact on all the sub-children. Thus not only will it result in impacting N5 but also nodes N6 and N7. It is important to note that such an eviction has less impact on RPL’s non-storing MOP i.e. in case of non-storing mode N5 might end up selecting alternate parent N8 and does not result in any additional control overhead for node N6 and N7.

Thus RPL’s non-storing MOP provides additional eviction flexibility for a neighbor management policy in terms evicting directly connected child entries.
2.3.4. NCE Reinforcement

It is expected that the latest reachability state and metric information be maintained in context to the NCE. With wireless networks, the neighbor cache entries prioritization may change over a period of time especially the link quality estimation parameters or the routing metrics. Reinforcement refers to updating the parameters in context to the NCEs which helps in prioritizing the entries when it comes to handling eviction. In wireless networks, on reception of incoming packet, the receiver node’s physical and MAC layer may derive certain signal reception parameters (such as RSSI, LQI) which can be considered for reinforcement purpose if the corresponding transmitter/source entry in neighbor cache is found. It should be noted that the signal quality parameters may have high variance in 6lo networks and thus statistical techniques (such as weighted averaging) are usually employed for deciding about a link quality over a period of time. Reinforcement can be achieved using one or more of the following techniques:

Passive Monitoring: Reinforcing the quality parameters using packets received from the source. Trickled DIO, periodic beacons, application traffic etc can be used for such monitoring.

Active Probing: A node may select subset of entries for active probing wherein it sends a message to the neighbor entry’s target and can expect a response message back. An example of such probing is [CONTIKI] where unicast DIS is sent to solicit a unicast DIO without impacting the trickle timers. Though it adds a control overhead on the link, periodic probing can help to ascertain connectivity in the absence of any other traffic from the neighboring node.

2.4. Requirements of a good neighbor management policy

Route Stability: Stable NCEs will result in stable routing adjacencies. Thus it is important to avoid unnecessary NCE churn for routing path stability.

Control overhead: A neighbor management policy may have to use signalling messages for policy handling (such as rejection of NCE). It is required that such overhead be kept as low as possible.

Scalability: Scalability with respect to large and uneven node densities in the network.
2.5. Approaches to neighbor management policy

Neighbor management policy depends upon the neighbor cache space availability and the same can be advertised proactively or can be handled reactively.

2.5.1. Reactive Approach

In this approach, the nodes select their RPL parent or the relay element purely based on link metrics and subsequently when they try to allocate their NCE in the target node, it may fail due to unavailability of the cache space. The failure can be communicated depending upon the signaling involved:

NS failure: Section 6.5.3 of [RFC6775] defines a procedure for NS failure handling in case the router’s neighbor cache is full. It results in a unicast NA with ARO status field set to two.

DAO NACK: Section 9.3 of RPL [RFC6550] specifies on how can the parent node react to DAOs from child. In case the parent could not make a NCE on behalf of the child node, a negative ACK with status (between 127-255) should be sent to the child node. The natural reaction of the child node would be to switch to an alternate parent.

PANA Failure: PaC’s auth session starts with a PaC discovering a PRE. The discovery procedure is not standardized and can be based upon various factors including signal strength of discovery messages from PRE. Post discovery, the PaC needs to send an unsecured unicast NS message with an ARO containing its link-local IPv6 address. NS helps to determine whether the PRE can allocate an NCE for the PaC. PRE accordingly sends a NA response with appropriate status field.

2.5.2. Proactive Approach

Neighbor cache availability could be proactively advertised by the parent nodes in the DIO messages and in the PRE discovery messages. A child RPL node may additionally use this information from DIO as part of parent selection process. In case of new joinee node, the node may use PRE discovery messages with space availability information to select an appropriate PRE. Proactive signaling of neighbor cache space availability will help the nodes to select the parent node or relay node such that the failure signaling due to cache full event can be reduced.

Currently there is no standard way of signaling such neighbor cache space availability information. RPL’s DIO messages carry metric
information and can be augmented with neighbor cache space as an additional metric. In case of PRE discovery however there is no standard way of defining this information since the PRE discovery procedure itself is not standardized.

In a wireless or shared bus network, a multicast DIO metric advertisement may reach several child nodes eventually everyone responding by selecting the same parent node causing neighbor cache to be exhausted. Thus the failure handling approaches defined in the Reactive Approach section applies here as well. But importantly the failure signaling will be significantly reduced because of proactive advertisement.

3. Reservation based Neighbor Management Policy

This section defines a sample neighbor management policy, with the primary objective to reduce NCE churn and to ensure stability of routing adjacencies. The scheme uses a reservation based policy to reserve NCEs for:

<table>
<thead>
<tr>
<th>NCE Entry for</th>
<th>MAX count</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing Parent</td>
<td>MAX_ROUTING_PARENT_NCE_NUM</td>
<td>PARENT</td>
</tr>
<tr>
<td>Routing child</td>
<td>MAX_ROUTING_CHILD_NCE_NUM</td>
<td>CHILD</td>
</tr>
<tr>
<td>Others such as pre-auth sessions</td>
<td>MAX_OTHER_NCE_NUM</td>
<td>OTHER</td>
</tr>
</tbody>
</table>

Table 1: Neighbor Cache Entry reservation

Note that reservation policy depends upon identification of the reason behind making an NCE. In case of pre-auth sessions, the corresponding NCE is created based on the unsecured NS/NA. In case of storing MOP, CHILD_ENT NCEs are created either based on DAO (as shown in Figure 3) or based on secured NS/NA messaging (as shown in Figure 4). In case of non-storing MOP, a secured NS/NA messaging as shown in Figure 4 needs to be used.

<- - - - - - - - - - - Routing Parents - - - - - - - - - - - ->
+---------------------------------------------------------------+
|                                 |                   |         |
|-----------------------------------------------------+---------|
| Routing Child                    Routing Parents     Other |

Figure 5: Reservation of NCEs in neighbor table
As shown in the figure, the neighbor cache is partitioned into different entry types. The routing parents can possibly occupy any entry type if found vacant since in case an eviction is sought the non-preferred routing parent could be evicted without much impact on the functioning or on the control traffic. The eviction could be done based on reasons specified in Section 2.3.3.

Routing Child entries are made in context to directly connected peers and these entries are not deleted unless they are unreachable or there is any reason for the parent node to believe that it is no longer the preferred parent for the child node. Deletion may happen based on reasons mentioned in Section 2.3.2.

Other entries (OTHER) may be made in response to temporary requirement of making an NCE. One such case is the pre-authentication phase where in the relay node makes an entry of the PaC temporarily till the time the authentication phase is completed. The NCE made thus is garbage collected at the end of the lifetime. Also an implementation may choose to keep a lower lifetime for such NCEs depending upon the time taken to complete the authentication process.

3.1. Limitations of such a policy

The reservation based policy mentioned in this section may result in sub-optimal path selection due to lack of NCE resource on the parent nodes. Also the restriction of maximum pre-auth sessions in the form of MAX_OTHER_NCE_NUM limits the maximum relay sessions that can be supported on the relay node.

The reservation policy allows the parent node to reject the child node’s DAO or NS. But the child node cannot remember this rejection and may reattempt the same parent after some time depending upon triggers such as reception of DIO from the same parent who rejected it previously. One of the only way to stop the child node from reattempting such parent selection would be to also include a proactive approach wherein the parent node signals its resource availability in the DIO message as mentioned in Section 2.5.2. Such a scheme of signalling parent node’s resource availability is currently not standardized.

RPL’s storing MOP imposes additional restrictions. One such case is where a child node may have a given parent node as its only parent and that parent node’s NCE are all used up. In such a case, the child node would keep on retrying and failing to send a DAO through the parent node. Ideally the parent node could have evicted a least used child node or a child node who has an alternate parent available. Evicting such a child node is a complex process and may
increase the control overhead as described in Section 2.3.3.1. Thus the reservation based policy requires that the minimum node density is sufficiently high so that every child finds a parent node in its vicinity with enough resources.

4. Acknowledgements

Thanks to Malisa Vucinic for pointing towards security aspects of un-authenticated nodes neighbor cache management.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

The Join Relay or the PANA Relay Element allows the un-authenticated nodes to join the network. Since the NS/NA signaling is unencrypted, it is possible that a malicious node may try spoof and occupy all the entries. This draft explains the use of reservation based policy for managing such entries. Following points try to reduce the impact of such attacks: Only a subset of entries are reserved for un-authenticated nodes.

a. Only a subset of entries are reserved for un-authenticated nodes doing plain-text NS/NA

b. It is recommended that NCE timeout be configured a lower value to evict such entries as soon as possible. New joining nodes are supposed to use these entries and thus are only needed during the time the authentication is in progress. Thus a short-duration NCE timeout will help reduce the impact of DoS attacks.

7. References

7.1. Normative References


7.2. Informative References


Authors’ Addresses
TCP Usage Guidance in the Internet of Things (IoT)
draft-ietf-lwig-tcp-constrained-node-networks-03

Abstract

This document provides guidance on how to implement and use the Transmission Control Protocol (TCP) in Constrained-Node Networks (CNNs), which are a characteristic of the Internet of Things (IoT). Such environments require a lightweight TCP implementation and may not make use of optional functionality. This document explains a number of known and deployed techniques to simplify a TCP stack as well as corresponding tradeoffs. The objective is to help embedded developers with decisions on which TCP features to use.

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 December 12, 2018.

Copyright Notice

Copyright (c) 2018 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.

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. Conventions used in this document ..................... 4
3. Characteristics of CNNs relevant for TCP .............. 4
   3.1. Network and link properties .................. 4
   3.2. Usage scenarios .............................. 5
   3.3. Communication and traffic patterns ........... 6
4. TCP implementation and configuration in CNNs .......... 6
   4.1. Path properties ............................... 7
   4.1.1. Maximum Segment Size (MSS) ............ 7
   4.1.2. Explicit Congestion Notification (ECN) . 7
   4.1.3. Explicit loss notifications ............. 8
   4.2. TCP guidance for small windows and buffers .... 8
   4.2.1. Single-MSS stacks - benefits and issues 8
   4.2.2. TCP options for single-MSS stacks ....... 9
   4.2.3. Delayed Acknowledgments for single-MSS stacks 9
   4.2.4. RTO estimation for single-MSS stacks ... 10
4.3. General recommendations for TCP in CNNs .......... 10
   4.3.1. Error recovery and congestion/flow control 10
   4.3.2. Selective Acknowledgments (SACK) ....... 11
   4.3.3. Delayed Acknowledgments .................. 11
5. TCP usage recommendations in CNNs .................... 11
   5.1. TCP connection initiation .................... 12
   5.2. TCP connection lifetime ..................... 12
      5.2.1. Long TCP connection lifetime .......... 12
      5.2.2. Short TCP connection lifetime ....... 12
   5.3. Number of parallel connections ............... 13
6. Security Considerations ................................ 13
7. Acknowledgments .......................................... 14
8. Annex. TCP implementations for constrained devices ... 14
   8.1. uIP ............................................ 14
   8.2. lwIP .......................................... 15
   8.3. RIOT .......................................... 15
   8.4. OpenWSN ...................................... 16
   8.5. TinyOS ........................................ 16
   8.6. FreeRTOS ....................................... 16
   8.7. uC/OS .......................................... 16
   8.8. Summary ........................................ 16
9. Annex. Changes compared to previous versions ........ 17
   9.1. Changes between -00 and -01 .................. 18
1. Introduction

The Internet Protocol suite is being used for connecting Constrained-Node Networks (CNNs) to the Internet, enabling the so-called Internet of Things (IoT) [RFC7228]. In order to meet the requirements that stem from CNNs, the IETF has produced a suite of new protocols specifically designed for such environments (see e.g. [I-D.ietf-lwig-energy-efficient]). New IETF protocol stack components include the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) adaptation layer, the IPv6 Routing Protocol for Low-power and lossy networks (RPL) routing protocol, and the Constrained Application Protocol (CoAP).

As of the writing, the main current transport layer protocols in IP-based IoT scenarios are UDP and TCP. However, TCP has been criticized (often, unfairly) as a protocol for the IoT. In fact, some TCP features are not optimal for IoT scenarios, such as relatively long header size, unsuitability for multicast, and always-confirmed data delivery. However, many typical claims on TCP unsuitability for IoT (e.g. a high complexity, connection-oriented approach incompatibility with radio duty-cycling, and spurious congestion control activation in wireless links) are not valid, can be solved, or are also found in well accepted IoT end-to-end reliability mechanisms (see [IntComp] for a detailed analysis).

At the application layer, CoAP was developed over UDP [RFC7252]. However, the integration of some CoAP deployments with existing infrastructure is being challenged by middleboxes such as firewalls, which may limit and even block UDP-based communications. This the main reason why a CoAP over TCP specification has been developed [RFC8323].

Other application layer protocols not specifically designed for CNNs are also being considered for the IoT space. Some examples include HTTP/2 and even HTTP/1.1, both of which run over TCP by default [RFC7540] [RFC2616], and the Extensible Messaging and Presence Protocol (XMPP) [RFC6120]. TCP is also used by non-IETF application-layer protocols in the IoT space such as the Message Queue Telemetry Transport (MQTT) and its lightweight variants.
TCP is a sophisticated transport protocol that includes optional functionality (e.g. TCP options) that may improve performance in some environments. However, many optional TCP extensions require complex logic inside the TCP stack and increase the codesize and the RAM requirements. Many TCP extensions are not required for interoperability with other standard-compliant TCP endpoints. Given the limited resources on constrained devices, careful "tuning" of the TCP implementation can make an implementation more lightweight.

This document provides guidance on how to implement and use TCP in CNNs. The overarching goal is to offer simple measures to allow for lightweight TCP implementation and suitable operation in such environments. A TCP implementation following the guidance in this document is intended to be compatible with a TCP endpoint that is compliant to the TCP standards, albeit possibly with a lower performance. This implies that such a TCP client would always be able to connect with a standard-compliant TCP server, and a corresponding TCP server would always be able to connect with a standard-compliant TCP client.

This document assumes that the reader is familiar with TCP. A comprehensive survey of the TCP standards can be found in [RFC7414]. Similar guidance regarding the use of TCP in special environments has been published before, e.g., for cellular wireless networks [RFC3481].

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 [RFC2119].

3. Characteristics of CNNs relevant for TCP

3.1. Network and link properties

CNNs are defined in [RFC7228] as networks whose characteristics are influenced by being composed of a significant portion of constrained nodes. The latter are characterized by significant limitations on processing, memory, and energy resources, among others [RFC7228]. The first two dimensions pose constraints on the complexity and on the memory footprint of the protocols that constrained nodes can support. The latter requires techniques to save energy, such as radio duty-cycling in wireless devices [I-D.ietf-lwig-energy-efficient], as well as minimization of the number of messages transmitted/received (and their size).
[RFC7228] lists typical network constraints in CNN, including low achievable bitrate/throughput, high packet loss and high variability of packet loss, highly asymmetric link characteristics, severe penalties for using larger packets, limits on reachability over time, etc. CNN may use wireless or wired technologies (e.g., Power Line Communication), and the transmission rates are typically low (e.g. below 1 Mbps).

For use of TCP, one challenge is that not all technologies in CNN may be aligned with typical Internet subnetwork design principles [RFC3819]. For instance, constrained nodes often use physical/link layer technologies that have been characterized as ‘lossy’, i.e., exhibit a relatively high bit error rate. Dealing with corruption loss is one of the open issues in the Internet [RFC6077].

3.2. Usage scenarios

There are different deployment and usage scenarios for CNNs. Some CNNs follow the star topology, whereby one or several hosts are linked to a central device that acts as a router connecting the CNN to the Internet. CNNs may also follow the multihop topology [RFC6606]. One key use case for the use of TCP is a model where constrained devices connect to unconstrained servers in the Internet. But it is also possible that both TCP endpoints run on constrained devices.

In constrained environments, there can be different types of devices [RFC7228]. For example, there can be devices with single combined send/receive buffer, devices with a separate send and receive buffer, or devices with a pool of multiple send/receive buffers. In the latter case, it is possible that buffers also be shared for other protocols.

When a CNN comprising one or more constrained devices and an unconstrained device communicate over the Internet using TCP, the communication possibly has to traverse a middlebox (e.g. a firewall, NAT, etc.). Figure 1 illustrates such scenario. Note that the scenario is asymmetric, as the unconstrained device will typically not suffer the severe constraints of the constrained device. The unconstrained device is expected to be mains-powered, to have high amount of memory and processing power, and to be connected to a resource-rich network.

Assuming that a majority of constrained devices will correspond to sensor nodes, the amount of data traffic sent by constrained devices (e.g. sensor node measurements) is expected to be higher than the amount of data traffic in the opposite direction. Nevertheless, constrained devices may receive requests (to which they may respond),
commands (for configuration purposes and for constrained devices including actuators) and relatively infrequent firmware/software updates.

Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

3.3. Communication and traffic patterns

IoT applications are characterized by a number of different communication patterns. The following non-comprehensive list explains some typical examples:

- Unidirectional transfers: An IoT device (e.g. a sensor) can send (repeatedly) updates to the other endpoint. Not in every case there is a need for an application response back to the IoT device.

- Request-response patterns: An IoT device receiving a request from the other endpoint, which triggers a response from the IoT device.

- Bulk data transfers: A typical example for a long file transfer would be an IoT device firmware update.

A typical communication pattern is that a constrained device communicates with an unconstrained device (cf. Figure 1). But it is also possible that constrained devices communicate amongst themselves.

4. TCP implementation and configuration in CNNs

This section explains how a TCP stack can deal with typical constraints in CNN. The guidance in this section relates to the TCP implementation and its configuration.
4.1. Path properties

4.1.1. Maximum Segment Size (MSS)

Some link layer technologies in the CNN space are characterized by a short data unit payload size, e.g. up to a few tens or hundreds of bytes. For example, the maximum frame size in IEEE 802.15.4 is 127 bytes. 6LoWPAN defined an adaptation layer to support IPv6 over IEEE 802.15.4 networks. The adaptation layer includes a fragmentation mechanism, since IPv6 requires the layer below to support an MTU of 1280 bytes [RFC2460], while IEEE 802.15.4 lacked fragmentation mechanisms. 6LoWPAN defines an IEEE 802.15.4 link MTU of 1280 bytes [RFC4944]. Other technologies, such as Bluetooth LE [RFC7668], ITU-T G.9959 [RFC7428] or DECT-ULE [RFC8105], also use 6LoWPAN-based adaptation layers in order to enable IPv6 support. These technologies do support link layer fragmentation. By exploiting this functionality, the adaptation layers that enable IPv6 over such technologies also define an MTU of 1280 bytes.

On the other hand, there exist technologies also used in the CNN space, such as Master Slave / Token Passing (TP) [RFC8163], Narrowband IoT (NB-IoT) [I-D.ietf-lpwan-overview] or IEEE 802.11ah [I-D.delpcarpio-6lo-wlanah], that do not suffer the same degree of frame size limitations as the technologies mentioned above. The MTU for MS/TP is recommended to be 1500 bytes [RFC8163], the MTU in NB-IoT is 1600 bytes, and the maximum frame payload size for IEEE 802.11ah is 7991 bytes.

For the sake of lightweight implementation and operation, unless applications require handling large data units (i.e. leading to an IPv6 datagram size greater than 1280 bytes), it may be desirable to limit the MTU to 1280 bytes in order to avoid the need to support Path MTU Discovery [RFC1981].

An IPv6 datagram size exceeding 1280 bytes can be avoided by setting the TCP MSS not larger than 1220 bytes. (Note: IP version 6 is assumed.)

4.1.2. Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) [RFC3168] has a number of benefits that are relevant for CNNs. ECN allows a router to signal in the IP header of a packet that congestion is arising, for example when a queue size reaches a certain threshold. An ECN-enabled TCP receiver will echo back the congestion signal to the TCP sender by setting a flag in its next TCP ACK. The sender triggers congestion control measures as if a packet loss had happened. ECN can be incrementally deployed in the Internet. Guidance on configuration
Internet-Draft                 TCP in IoT                      June 2018

and usage of ECN is provided in [RFC7567]. The document [RFC8087] outlines the principal gains in terms of increased throughput, reduced delay, and other benefits when ECN is used over a network path that includes equipment that supports Congestion Experienced (CE) marking.

ECN can reduce packet losses since congestion control measures can be applied earlier [RFC2884]. Less lost packets implies that the number of retransmitted segments decreases, which is particularly beneficial in CNNs, where energy and bandwidth resources are typically limited. Also, it makes sense to try to avoid packet drops for transactional workloads with small data sizes, which are typical for CNNs. In such traffic patterns, it is more difficult to detect packet loss without retransmission timeouts (e.g., as there may be no three duplicate ACKs). Any retransmission timeout slows down the data transfer significantly. When the congestion window of a TCP sender has a size of one segment, the TCP sender resets the retransmit timer, and the sender will only be able to send a new packet when the retransmit timer expires [RFC3168]. Effectively, the TCP sender reduces at that moment its sending rate from 1 segment per Round Trip Time (RTT) to 1 segment per RTO, which can result in a very low throughput. In addition to better throughput, ECN can also help reducing latency and jitter.

Given the benefits, more and more TCP stacks in the Internet support ECN, and it specifically makes sense to leverage ECN in controlled environments such as CNNs.

4.1.3. Explicit loss notifications

There has been a significant body of research on solutions capable of explicitly indicating whether a TCP segment loss is due to corruption, in order to avoid activation of congestion control mechanisms [ETEN] [RFC2757]. While such solutions may provide significant improvement, they have not been widely deployed and remained as experimental work. In fact, as of today, the IETF has not standardized any such solution.

4.2. TCP guidance for small windows and buffers

This section discusses TCP stacks that focus on transferring a single MSS. More general guidance is provided in Section 4.3.

4.2.1. Single-MSS stacks - benefits and issues

A TCP stack can reduce the RAM requirements by advertising a TCP window size of one MSS, and also transmit at most one MSS of unacknowledged data. In that case, both congestion and flow control
implementation is quite simple. Such a small receive and send window may be sufficient for simple message exchanges in the CNN space. However, only using a window of one MSS can significantly affect performance. A stop-and-wait operation results in low throughput for transfers that exceed the lengths of one MSS, e.g., a firmware download.

If CoAP is used over TCP with the default setting for NSTART in [RFC7252], a CoAP endpoint is not allowed to send a new message to a destination until a response for the previous message sent to that destination has been received. This is equivalent to an application-layer window size of 1. For this use of CoAP, a maximum TCP window of one MSS will be sufficient.

4.2.2. TCP options for single-MSS stacks

A TCP implementation needs to support options 0, 1 and 2 [RFC0793]. These options are sufficient for interoperability with a standard-compliant TCP endpoint, albeit many TCP stacks support additional options and can negotiate their use.

A TCP implementation for a constrained device that uses a single-MSS TCP receive or transmit window size may not benefit from supporting the following TCP options: Window scale [RFC1323], TCP Timestamps [RFC1323], Selective Acknowledgments (SACK) and SACK-Permitted [RFC2018]. Also other TCP options may not be required on a constrained device with a very lightweight implementation.

One potentially relevant TCP option in the context of CNNs is TCP Fast Open (TFO) [RFC7413]. As described in Section 5.2.2, TFO can be used to address the problem of traversing middleboxes that perform early filter state record deletion.

4.2.3. Delayed Acknowledgments for single-MSS stacks

TCP Delayed Acknowledgments are meant to reduce the number of transferred bytes within a TCP connection, but they may increase the time until a sender may receive an ACK. There can be interactions with stacks that use very small windows.

A device that advertises a single-MSS receive window should avoid use of delayed ACKs in order to avoid contributing unnecessary delay (of up to 500 ms) to the RTT [RFC5681], which limits the throughput and can increase the data delivery time.

A device that can send at most one MSS of data is significantly affected if the receiver uses delayed ACKs, e.g., if a TCP server or receiver is outside the CNN. One known workaround is to split the...
data to be sent into two segments of smaller size. A standard compliant TCP receiver will then immediately acknowledge the second segment, which can improve throughput. This "split hack" works if the TCP receiver uses Delayed Acks, but the downside is the overhead of sending two IP packets instead of one.

4.2.4. RTO estimation for single-MSS stacks

The Retransmission Timeout (RTO) estimation is one of the fundamental TCP algorithms. There is a fundamental trade-off: A short, aggressive RTO behavior reduces wait time before retransmissions, but it also increases the probability of spurious timeouts. The latter lead to unnecessary waste of potentially scarce resources in CNNs such as energy and bandwidth. In contrast, a conservative timeout can result in long error recovery times and thus needlessly delay data delivery.

[RFC6298] describes the standard TCP RTO algorithm. If a TCP sender uses very small window size and cannot use Fast Retransmit/Fast Recovery or SACK, the Retransmission Timeout (RTO) algorithm has a larger impact on performance than for a more powerful TCP stack. In that case, RTO algorithm tuning may be considered, although careful assessment of possible drawbacks is recommended.

As an example, an adaptive RTO algorithm for CoAP over UDP has been defined [I-D.ietf-core-cocoa] that has been found to perform well in CNN scenarios [Commag].

4.3. General recommendations for TCP in CNNs

This section summarizes some widely used techniques to improve TCP, with a focus on their use in CNNs. The TCP extensions discussed here are useful in a wide range of network scenarios, including CNNs. This section is not comprehensive. A comprehensive survey of TCP extensions is published in [RFC7414].

4.3.1. Error recovery and congestion/flow control

Devices that have enough memory to allow larger TCP window size can leverage a more efficient error recovery using Fast Retransmit and Fast Recovery [RFC5681]. These algorithms work efficiently for window sizes of at least 5 MSS: If in a given TCP transmission of segments 1,2,3,4,5, and 6 the segment 2 gets lost, the sender should get an acknowledgement for segment 1 when 3 arrives and duplicate acknowledgements when 4, 5, and 6 arrive. It will retransmit segment 2 when the third duplicate ack arrives. In order to have segment 2, 3, 4, 5, and 6 sent, the window has to be at least five. With an MSS of 1220 byte, a buffer of the size of 5 MSS would require 6100 byte.
For bulk data transfers further TCP improvements may also be useful, such as limited transmit [RFC3042].

4.3.2. Selective Acknowledgments (SACK)

If a device with less severe memory and processing constraints can afford advertising a TCP window size of several MSSs, it makes sense to support the SACK option to improve performance. SACK allows a data receiver to inform the data sender of non-contiguous data blocks received, thus a sender (having previously sent the SACK-Permitted option) can avoid performing unnecessary retransmissions, saving energy and bandwidth, as well as reducing latency. SACK is particularly useful for bulk data transfers. The receiver supporting SACK will need to manage the reception of possible out-of-order received segments, requiring sufficient buffer space. SACK adds $8n+2$ bytes to the TCP header, where $n$ denotes the number of data blocks received, up to 4 blocks. For a low number of out-of-order segments, the header overhead penalty of SACK is compensated by avoiding unnecessary retransmissions.

4.3.3. Delayed Acknowledgments

For certain traffic patterns, Delayed Acknowledgements may have a detrimental effect, as already noted in Section 4.2.3. Advanced TCP stacks may use heuristics to determine the maximum delay for an ACK. For CNNs, the recommendation depends on the expected communication patterns.

If a stack is able to deal with more than one MSS of data, it may make sense to use a small timeout or disable delayed ACKs when traffic over a CNN is expected to mostly be small messages with a size typically below one MSS. For request-response traffic between a constrained device and a peer (e.g. backend infrastructure) that uses delayed ACKs, the maximum ACK rate of the peer will be typically of one ACK every 200 ms (or even lower). If in such conditions the peer device is administered by the same entity managing the constrained device, it is recommended to disable delayed ACKs at the peer side.

In contrast, delayed ACKs allow to reduce the number of ACKs in bulk transfer type of traffic, e.g. for firmware/software updates or for transferring larger data units containing a batch of sensor readings.

5. TCP usage recommendations in CNNs

This section discusses how a TCP stack can be used by applications that are developed for CNN scenarios. These remarks are by and large independent of how TCP is exactly implemented.
5.1. TCP connection initiation

In the constrained device to unconstrained device scenario illustrated above, a TCP connection is typically initiated by the constrained device, in order for this device to support possible sleep periods to save energy.

5.2. TCP connection lifetime

[[TODO: This section may need rewording in the next revision.]]

5.2.1. Long TCP connection lifetime

In CNNs, in order to minimize message overhead, a TCP connection should be kept open as long as the two TCP endpoints have more data to exchange or it is envisaged that further segment exchanges will take place within an interval of two hours since the last segment has been sent. A greater interval may be used in scenarios where applications exchange data infrequently.

TCP keep-alive messages [RFC1122] may be supported by a server, to check whether a TCP connection is active, in order to release state of inactive connections. This may be useful for servers running on memory-constrained devices.

Since the keep-alive timer may not be set to a value lower than two hours [RFC1122], TCP keep-alive messages are not useful to guarantee that filter state records in middleboxes such as firewalls will not be deleted after an inactivity interval typically in the order of a few minutes [RFC6092]. In scenarios where such middleboxes are present, alternative measures to avoid early deletion of filter state records (which might lead to frequent establishment of new TCP connections between the two involved endpoints) include increasing the initial value for the filter state inactivity timers (if possible), and using application layer heartbeat messages.

5.2.2. Short TCP connection lifetime

A different approach to addressing the problem of traversing middleboxes that perform early filter state record deletion relies on using TFO [RFC7413]. In this case, instead of trying to maintain a TCP connection for long time, possibly short-lived connections can be opened between two endpoints while incurring low overhead. In fact, TFO allows data to be carried in SYN (and SYN-ACK) packets, and to be consumed immediately by the receiving endpoint, thus reducing overhead compared with the traditional three-way handshake required to establish a TCP connection.
For security reasons, TFO requires the TCP endpoint that will open the TCP connection (which in CNNs will typically be the constrained device) to request a cookie from the other endpoint. The cookie, with a size of 4 or 16 bytes, is then included in SYN packets of subsequent connections. The cookie needs to be refreshed (and obtained by the client) after a certain amount of time. Nevertheless, TFO is more efficient than frequently opening new TCP connections (by using the traditional three-way handshake) for transmitting new data, as long as the cookie update rate is well below the data new connection rate.

5.3. Number of parallel connections

[[TODO: This has been added in -02 but needs further alignment]]

TCP endpoints with a small amount of RAM may only support a small number of connections. Each connection may result in overhead, and depending on the internal TCP implementation, they may compete for scarce resources. A careful application design may try to keep the number of parallel connections as small as possible.

6. Security Considerations

Best current practice for securing TCP and TCP-based communication also applies to CNN. As example, use of Transport Layer Security (TLS) is strongly recommended if it is applicable.

There are also TCP options which can improve TCP security. Examples include the TCP MD5 signature option [RFC2385] and the TCP Authentication Option (TCP-AO) [RFC5925]. However, both options add overhead and complexity. The TCP MD5 signature option adds 18 bytes to every segment of a connection. TCP-AO typically has a size of 16-20 bytes.

For the mechanisms discussed in this document, the corresponding considerations apply. For instance, if TFO is used, the security considerations of [RFC7413] apply.

Constrained devices are expected to support smaller TCP window sizes than less limited devices. In such conditions, segment retransmission triggered by RTO expiration is expected to be relatively frequent, due to lack of (enough) duplicate ACKs, especially when a constrained device uses a single-MSS window size. For this reason, constrained devices running TCP may appear as particularly appealing victims of the so-called "shrew" Denial of Service (DoS) attack [shrew], whereby one or more sources generate a packet spike targeted to coincide with consecutive RTO-expiration-triggered retry attempts of a victim node. Note that the attack may
be performed by Internet-connected devices, including constrained devices in the same CNN as the victim, as well as remote ones. Mitigation techniques include RTO randomization and attack blocking by routers able to detect shrew attacks based on their traffic pattern.

7. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

The authors appreciate the feedback received for this document. The following folks provided comments that helped improve the document: Carsten Bormann, Zhen Cao, Wei Genyu, Ari Keranen, Abhijan Bhattacharyya, Andres Arcia-Moret, Yoshifumi Nishida, Joe Touch, Fred Baker, Nik Sultana, Kerry Lynn, Erik Nordmark, Markku Kojo, and Hannes Tschofenig. Simon Brummer provided details, and kindly performed RAM and ROM usage measurements, on the RIOT TCP implementation. Xavi Vilajosana provided details on the OpenWSN TCP implementation. Rahul Jadhav provided details on the uIP TCP implementation.

8. Annex. TCP implementations for constrained devices

This section overviews the main features of TCP implementations for constrained devices. The survey is limited to open source stacks with small footprint. It is not meant to be all-encompassing. For more powerful embedded systems (e.g., with 32-bit processors), there are further stacks that comprehensively implement TCP. On the other hand, please be aware that this Annex is based on information available as of the writing.

8.1. uIP

uIP is a TCP/IP stack, targeted for 8 and 16-bit microcontrollers, which pioneered TCP/IP implementations for constrained devices. uIP has been deployed with Contiki and the Arduino Ethernet shield. A code size of ~5 kB (which comprises checksumming, IP, ICMP and TCP) has been reported for uIP [Dunk].

uIP uses same buffer both incoming and outgoing traffic, with has a size of a single packet. In case of a retransmission, an application
must be able to reproduce the same user data that had been transmitted.

The MSS is announced via the MSS option on connection establishment and the receive window size (of one MSS) is not modified during a connection. Stop-and-wait operation is used for sending data. Among other optimizations, this allows to avoid sliding window operations, which use 32-bit arithmetic extensively and are expensive on 8-bit CPUs.

Contiki uses the "split hack" technique (see Section 4.2.3) to avoid delayed ACKs for senders using a single MSS.

8.2. lwIP

lwIP is a TCP/IP stack, targeted for 8- and 16-bit microcontrollers. lwIP has a total code size of \( \sim 14 \text{ kB to } \sim 22 \text{ kB} \) (which comprises memory management, checksumming, network interfaces, IP, ICMP and TCP), and a TCP code size of \( \sim 9 \text{ kB to } \sim 14 \text{ kB} \) [Dunk].

In contrast with uIP, lwIP decouples applications from the network stack. lwIP supports a TCP transmission window greater than a single segment, as well as buffering of incoming and outgoing data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit and fast recovery. SACK and Window Scale have been recently added to lwIP.

8.3. RIOT

The RIOT TCP implementation (called GNRC TCP) has been designed for Class 1 devices [RFC 7228]. The main target platforms are 8- and 16-bit microcontrollers. GNRC TCP offers a similar function set as uIP, but it provides and maintains an independent receive buffer for each connection. In contrast to uIP, retransmission is also handled by GNRC TCP. GNRC TCP uses a single-MSS window size, which simplifies the implementation. The application programmer does not need to know anything about the TCP internals, therefore GNRC TCP can be seen as a user-friendly uIP TCP implementation.

The MSS is set on connections establishment and cannot be changed during connection lifetime. GNRC TCP allows multiple connections in parallel, but each TCB must be allocated somewhere in the system. By default there is only enough memory allocated for a single TCP connection, but it can be increased at compile time if the user needs multiple parallel connections.
The RIOT TCP implementation does not currently support classic POSIX sockets. However, it supports an interface that has been inspired by POSIX.

8.4. OpenWSN

The TCP implementation in OpenWSN has similar functionality like the uIP TCP implementation. OpenWSN TCP implementation only supports the minimum state machine functionality required. For example, it does not perform retransmissions. There has not been much recent activity to overcome these limitations.

8.5. TinyOS

TinyOS was important as platform for early constrained devices. TinyOS has an experimental TCP stack that uses a simple nonblocking library-based implementation of TCP, which provides a subset of the socket interface primitives. The application is responsible for buffering. The TCP library does not do any receive-side buffering. Instead, it will immediately dispatch new, in-order data to the application and otherwise drop the segment. A send buffer is provided so that the TCP implementation can automatically retransmit missing segments. Multiple TCP connections are possible. Recently there has been little further work on the stack.

8.6. FreeRTOS

FreeRTOS is a real-time operating system kernel for embedded devices that is supported by 16- and 32-bit microprocessors. Its TCP implementation is based on multiple-MSS window size, although a ‘Tiny-TCP’ option, which is a single-MSS variant, can be enabled. Delayed ACKs are supported, with a 20-ms Delayed ACK timer as a technique intended ‘to gain performance’.

8.7. uC/OS

uC/OS is a real-time operating system kernel for embedded devices, which is maintained by Micrium. uC/OS is intended for 8-, 16- and 32-bit microprocessors. The uC/OS TCP implementation supports a multiple-MSS window size.

8.8. Summary
<table>
<thead>
<tr>
<th>OS</th>
<th>Memory</th>
<th>Code size (kB)</th>
<th>Win size (MSS)</th>
<th>Slow start</th>
<th>Fast rec/retx</th>
<th>Keep-alive</th>
<th>Win. Scale</th>
<th>TCP timest.</th>
<th>SACK</th>
<th>Del. ACKs</th>
<th>Concur. Conn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>uIP</td>
<td>&lt;5</td>
<td>&lt;5</td>
<td>Mult.</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>lwIP orig</td>
<td>9 to 14</td>
<td>40</td>
<td>Mult.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>lwIP 2.0</td>
<td>7</td>
<td>&lt;7</td>
<td>1</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RIOT</td>
<td>40</td>
<td>1</td>
<td>Mult.</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OpenWSN</td>
<td>&lt;9.2</td>
<td>N/A</td>
<td>Mult.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TinyOS</td>
<td>N/A</td>
<td>N/A</td>
<td>Mult.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>&lt;9.2</td>
<td>N/A</td>
<td>Mult.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>uC/OS</td>
<td>N/A</td>
<td>N/A</td>
<td>Mult.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- (T1) = TCP-only, on x86 and AVR platforms
- (T2) = TCP-only, on ARM Cortex-M platform
- (T3) = TCP-only, on ARM Cortex-M0+ platform (NOTE: RAM usage for the same platform is ~2.5 kB for one TCP connection plus ~1.2 kB for each additional connection)
- (a) = includes IP, ICMP and TCP on x86 and AVR platforms
- (b) = the whole protocol stack on mbed
- (I) = interface inspired by POSIX
- Mult. = Multiple
N/A = Not Available

Figure 2: Summary of TCP features for different lightweight TCP implementations. None of the implementations considered in this Annex support ECN or TFO.

9. Annex. Changes compared to previous versions

RFC Editor: To be removed prior to publication
9.1. Changes between -00 and -01
- Changed title and abstract
- Clarification that communication with standard-compliant TCP endpoints is required, based on feedback from Joe Touch
- Additional discussion on communication patterns
- Numerous changes to address a comprehensive review from Hannes Tschofenig
- Reworded security considerations
- Additional references and better distinction between normative and informative entries
- Feedback from Rahul Jadhav on the uIP TCP implementation
- Basic data for the TinyOS TCP implementation added, based on source code analysis

9.2. Changes between -01 and -02
- Added text to the Introduction section, and a reference, on traditional bad perception of TCP for IoT
- Added sections on FreeRTOS and uC/OS
- Updated TinyOS section
- Updated summary table
- Reorganized Section 4 (single-MSS vs multiple-MSS window size), some content now also in new Section 5

9.3. Changes between -02 and -03
- Rewording to better explain the benefit of ECN
- Additional context information on the surveyed implementations
- Added details, but removed "Data size" raw, in the summary table
- Added discussion on shrew attacks
10. References

10.1. Normative References


10.2. Informative References


[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,

[I-D.ietf-lpwan-overview]

[I-D.ietf-lwig-energy-efficient]
Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, "Energy-Efficient Features of Internet of Things Protocols",
draft-ietf-lwig-energy-efficient-08 (work in progress), October 2017.


Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu
Abstract

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP. The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, and OSCORE. DTLS and TLS are analyzed with and without 6LoWPAN-GHC compression. DTLS is analyzed with and without Connection ID.

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 August 26, 2018.

Copyright Notice

Copyright (c) 2018 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.
1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UDP [RFC7252] and TCP [RFC8323]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.ietf-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [I-D.ietf-tls-tls13], and OSCORE [I-D.ietf-core-object-security]. The DTLS and TLS record layers are analyzed with and without compression. DTLS is analyzed with and without Connection ID [I-D.ietf-tls-dtls-connection-id] and DTLS 1.3 is analyzed with and without the use of the short header. Readers are expected to be familiar with some of the terms described in RFC 7925 [RFC7925], such as ICV.
2. Overhead of Security Protocols

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (AES_128_CCM_8), a plaintext of 6 bytes, and the sequence number ‘05’. This follows the example in [RFC7400], Figure 16.

Note that the compressed overhead calculations for DLTS 1.2, DTLS 1.3, TLS 1.2 and TLS 1.3 are dependent on the parameters epoch, sequence number, and length, and all the overhead calculations are dependent on the parameter Connection ID when used. Note that the OSCORE overhead calculations are dependent on the CoAP option numbers, as well as the length of the OSCORE parameters Sender ID and Sequence Number. The following are only examples.

2.1. DTLS 1.2

2.1.1. DTLS 1.2

This section analyzes the overhead of DTLS 1.2 [RFC6347]. The nonce follow the strict profiling given in [RFC7925]. This example is taken directly from [RFC7400], Figure 16.

DTLS 1.2 Record Layer (35 bytes, 29 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 00 05 00 16 00 01 00
00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Cipheredtext:
ae a0 15 56 67 92
IV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 gives 29 bytes overhead.
2.1.2. DTLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with [RFC7400]. The compression was done with [OlegHahm-ghc].

Note that the sequence number ‘01’ used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 Record Layer (22 bytes, 16 bytes overhead):
\[
\text{Compressed DTLS 1.2 Record Layer Header and Nonce:}
\]
\[
b0 \text{ c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff}
\]
\[
8a 24 e4 cb 35 b9
\]

Ciphertext:
\[
ae a0 15 56 67 92
\]

ICV:
\[
4d ff 8a 24 e4 cb 35 b9
\]

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, length) gives 16 bytes overhead.

2.1.3. DTLS 1.2 with Connection ID

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id]. The overhead calculations in this section uses Connection ID = ‘42’. DTLS with a Connection ID = ‘’ (the empty string) is equal to DTLS without Connection ID.
DTLS 1.2 Record Layer (36 bytes, 30 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 42 00 16 00 01
00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Connection ID:
42
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 with Connection ID gives 30 bytes overhead.

2.1.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the sequence number ‘01’ used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.
Compressed DTLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
b0 c3 04 05 42 00 16 f2 0e ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.2 Record Layer Header and Nonce:
b0 c3 04 05 42 00 16 f2 0e
Ciphertext:
  ae a0 15 56 67 92
  4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters
(epoch, sequence number, Connection ID, length) gives 17 bytes
overhead.

2.2.  DTLS 1.3

2.2.1.  DTLS 1.3

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13].
The changes compared to DTLS 1.2 are: omission of version number,
merging of epoch and sequence number fields (of total 8 bytes) into
one 4-bytes-field.

DTLS 1.3 Record Layer (22 bytes, 16 bytes overhead):
17 40 00 00 05 00 0f ae a0 15 56 67 92 ec 4d ff
8a 24 e4 cb 35 b9

Content type:
  17
Epoch and Sequence:
  40 00 00 05
Length:
  00 0f
Ciphertext (including encrypted ContentType):
  ae a0 15 56 67 92 ec
  4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 16 bytes overhead.

2.2.2.  DTLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13]
when compressed with [RFC7400] [OlegHahm-ghc].
Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
02 17 40 80 12 05 00 0f ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
02 17 40 80 12 05 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.2.3. DTLS 1.3 with Connection ID

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id].

DTLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
17 40 00 00 05 42 00 0f ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Epoch and Sequence:
40 00 00 05
Connection ID:
42
Length:
00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 17 bytes overhead.

2.2.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with [RFC7400] [OlegHahm-ghc].
Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (24 bytes, 18 bytes overhead):
02 17 40 80 13 05 42 00 0f ae a0 15 56 67 92 ec
4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
02 17 40 80 13 05 42 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, Connection ID, length) gives 18 bytes overhead.

2.2.5. DTLS 1.3 with short header

This section analyzes the overhead of DTLS 1.3 with short header format [I-D.ietf-tls-dtls13]. The short header format for DTLS 1.3 reduces the header of 5 bytes, by omitting the length value and sending 1 lower bit of epoch value instead of 2, and 12 lower bits of sequence number instead of 30.

DTLS 1.3 Record Layer (17 bytes, 11 bytes overhead):
30 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 short header:
30 05
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 with short header gives 11 bytes overhead.

2.2.6. DTLS 1.3 with short header and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 with short header [I-D.ietf-tls-dtls13] when compressed with [RFC7400] [OlegHahm-ghc].
Compressed DTLS 1.3 Record Layer (18 bytes, 12 bytes overhead)
11 30 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 short header (including sequence number)
11 30 05
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 with short header gives 12 bytes overhead.

2.3. TLS 1.2

2.3.1. TLS 1.2

This section analyzes the overhead of TLS 1.2 [RFC5246]. The changes compared to DTLS 1.2 is that the TLS 1.2 record layer does not have epoch and sequence number, and that the version is different.

TLS 1.2 Record Layer (27 bytes, 21 bytes overhead):
17 03 03 00 16 00 00 00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 03
Length:
00 16
Nonce:
00 00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.2 gives 21 bytes overhead.

2.3.2. TLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.2 [RFC5246] when compressed with [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.
Compressed TLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
05 17 03 03 00 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.2 Record Layer Header and Nonce:
05 17 03 03 00 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.2 with the above parameters
(epoch, sequence number, length) gives 17 bytes overhead.

2.4. TLS 1.3

2.4.1. TLS 1.3

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13].
The change compared to TLS 1.2 is that the TLS 1.3 record layer uses
a different version.

TLS 1.3 Record Layer (20 bytes, 14 bytes overhead):
17 03 03 00 16 ae a0 15 56 67 92 ec 4d ff 8a 24
e4 cb 35 b9

Content type:
17
Legacy Version:
03 03
Length:
00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 14 bytes overhead.

2.4.2. TLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13]
when compressed with [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is
exchanged over transports that do not use 6LoWPAN together with
6LoWPAN-GHC.
Compressed TLS 1.3 Record Layer (21 bytes, 15 bytes overhead)
14 17 03 03 00 0f ae a0 15 56 67 92 ec 4d ff 8a
24 e4 cb 35 b9

Compressed TLS 1.3 Record Layer Header and Nonce:
14 17 03 03 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.3 with the above parameters
(epoch, sequence number, length) gives 15 bytes overhead.

2.5. OSCORE

This section analyzes the overhead of OSCORE
[I-D.ietf-core-object-security].

Note that Sender ID = ‘’ (empty string) can only be used by one
client per server.

The examples below assume that the original messages does not have
payload (note that this does not affect the overhead).

The below calculation Option Delta = ‘9’, Sender ID = ‘’ (empty
string), and Sequence Number = ‘05’, and is only an example.

OSCORE Request (19 bytes, 13 bytes overhead):
92 09 05
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
92
Option Value (flag byte and sequence number):
09 05
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation Option Delta = ‘9’, Sender ID = ‘42’, and
Sequence Number = ‘05’, and is only an example.
OSCORE Request (20 bytes, 14 bytes overhead):
93 09 05 42
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
93
Option Value (flag byte, sequence number, and Sender ID):
09 05 42
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation uses Option Delta = '9'.

OSCORE Response (17 bytes, 11 bytes overhead):
90
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Delta and Option Length:
90
Option Value
-
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCORE with the above parameters gives 13-14 bytes overhead for requests and 11 bytes overhead for responses.

Unlike DTLS and TLS, OSCORE has much smaller overhead for responses than requests.

3. Overhead with Different Parameters

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs with the same length.

The compression overhead (GHC) is dependent on the parameters epoch, sequence number, Connection ID, and length (where applicable). The following overheads should be representative for sequence numbers and Connection IDs with the same length.
The OSCORE overhead is dependent on the included CoAP Option numbers as well as the length of the OSCORE parameters Sender ID and sequence number. The following overheads apply for all sequence numbers and Sender IDs with the same length.

<table>
<thead>
<tr>
<th>Sequence Number</th>
<th>'05'</th>
<th>'1005'</th>
<th>'100005'</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS 1.2</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>DTLS 1.3</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>DTLS 1.3 (short header)</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>DTLS 1.2 (GHC)</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>DTLS 1.3 (GHC)</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>DTLS 1.3 (short header) (GCH)</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>TLS 1.2</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>TLS 1.3</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>TLS 1.2 (GHC)</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>TLS 1.3 (GHC)</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>OSCORE Request</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>OSCORE Response</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 1: Overhead in bytes as a function of sequence number (Connection/Sender ID = '')

<table>
<thead>
<tr>
<th>Connection/Sender ID</th>
<th>''</th>
<th>'42'</th>
<th>'4002'</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS 1.2</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>DTLS 1.3</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>DTLS 1.3 (short header)</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>DTLS 1.2 (GHC)</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>DTLS 1.3 (GHC)</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>DTLS 1.3 (short header) (GCH)</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>OSCORE Request</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>OSCORE Response</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 2: Overhead in bytes as a function of Connection/Sender ID (Sequence Number = '05')
DTLS 1.2                        21               8
DTLS 1.3                         8               9
DTLS 1.3 (short header)          3               4
---------------------------------------------------------------------
TLS  1.2                        13               9
TLS  1.3                         6               7
---------------------------------------------------------------------
OSCORE Request                   5
OSCORE Response                  3

Figure 3: Overhead (excluding ICV) in bytes       (Connection/Sender
ID = '', Sequence Number = '05')

4. Summary

DTLS 1.2 has quite a large overhead as it uses an explicit sequence
number and an explicit nonce. TLS 1.2 has significantly less (but
not small) overhead. TLS 1.3 and DTLS 1.3 have quite small overhead.
DTLS 1.3 with short header format has very small overhead.

The Generic Header Compression (6LoWPAN-GHC) can in addition to DTLS
1.2 handle TLS 1.2, and DTLS 1.2 with Connection ID. The Generic
Header Compression (6LoWPAN-GHC) works very well for Connection ID
and the overhead seems to increase exactly with the length of the
Connection ID (which is optimal). The compression of TLS 1.2 is not
as good as the compression of DTLS 1.2 (as the static dictionary only
contains the DTLS 1.2 version number). Similar compression levels as
for DTLS could be achieved also for TLS 1.2, but this would require
different static dictionaries. For TLS 1.3 and DTLS 1.3, GHC
increases the overhead. Note that GHC in some cases might be able to
compress the payload and therefore reduce total overhead.

The 6LoWPAN-GHC header compression is not available when (D)TLS is
exchanged over transports that do not use 6LoWPAN together with
6LoWPAN-GHC.

The short header format for DTLS 1.3 reduces the header of 5 bytes,
by omitting the length value and sending 1 lower bit of epoch value
instead of 2, and 12 lower bits of sequence number instead of 30.
This may create problems reconstructing the full sequence number, if
2000 datagrams in sequence are lost.

OSCORE has much lower overhead than DTLS (with no short header
format) and TLS. The overhead of OSCORE is smaller than DTLS over
6LoWPAN with compression, and this small overhead is achieved even on
deployments without 6LoWPAN or 6LoWPAN without DTLS compression.
OSCORE is lightweight because it makes use of some excellent features in CoAP, CBOR, and COSE.

5. Security Considerations

This document is purely informational.

6. IANA Considerations

This document has no actions for IANA.

7. Informative References

[I-D.ietf-core-object-security]

[I-D.ietf-tls-dtls-connection-id]

[I-D.ietf-tls-dtls13]

[I-D.ietf-tls-tls13]

[OlegHahm-ghc]


Acknowledgments

The authors want to thank Ari Keraenen, Carsten Bormann, Goeran Selander, and Hannes Tschofenig for comments and suggestions on previous versions of the draft.

All 6LoWPAN-GHC compression was done with [OlegHahm-ghc].

Authors’ Addresses

John Mattsson
Ericsson AB
Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Email: francesca.palombini@ericsson.com
Comparison of CoAP Security Protocols
draft-mattsson-lwig-security-protocol-comparison-01

Abstract

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP. The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, and OSCORE. DTLS and TLS are analyzed with and without 6LoWPAN-GHC compression. DTLS is analyzed with and without Connection ID.

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 September 20, 2018.

Copyright Notice

Copyright (c) 2018 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.
1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UDP [RFC7252] and TCP [RFC8323]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.ietf-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [I-D.ietf-tls-tls13], and OSCORE [I-D.ietf-core-object-security]. The DTLS and TLS record layers are analyzed with and without compression. DTLS is analyzed with and without Connection ID [I-D.ietf-tls-dtls-connection-id] and DTLS 1.3 is analyzed with and without the use of the short header. Readers are expected to be familiar with some of the terms described in RFC 7925 [RFC7925], such as ICV.
2. Overhead of Security Protocols

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (i.e. AES_128_CCM_8, AES-CCM-16-64, or AES-CCM-64-64), a plaintext of 6 bytes, and the sequence number '05'. This follows the example in [RFC7400], Figure 16.

Note that the compressed overhead calculations for DLTS 1.2, DTLS 1.3, TLS 1.2 and TLS 1.3 are dependent on the parameters epoch, sequence number, and length, and all the overhead calculations are dependent on the parameter Connection ID when used. Note that the OSCORE overhead calculations are dependent on the CoAP option numbers, as well as the length of the OSCORE parameters Sender ID and Sequence Number. The following are only examples.

2.1. DTLS 1.2

2.1.1. DTLS 1.2

This section analyzes the overhead of DTLS 1.2 [RFC6347]. The nonce follow the strict profiling given in [RFC7925]. This example is taken directly from [RFC7400], Figure 16.

DTLS 1.2 Record Layer (35 bytes, 29 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 00 16 00 01 00
00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 gives 29 bytes overhead.
2.1.2. DTLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with [RFC7400]. The compression was done with [OlegHahm-ghc].

Note that the sequence number ‘01’ used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 Record Layer (22 bytes, 16 bytes overhead):

b0 c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff
8a 24 e4 cb 35 b9

Compressed DTLS 1.2 Record Layer Header and Nonce:

b0 c3 03 05 00 16 f2 0e

Ciphertext:

ae a0 15 56 67 92

ICV:

4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, length) gives 16 bytes overhead.

2.1.3. DTLS 1.2 with Connection ID

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id]. The overhead calculations in this section uses Connection ID = ‘42’. DTLS with a Connection ID = ‘’ (the empty string) is equal to DTLS without Connection ID.
DTLS 1.2 Record Layer (36 bytes, 30 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 42 00 16 00 01
00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9

Content type: 17
Version: fe fd
Epoch: 00 01
Sequence number: 00 00 00 00 00 05
Connection ID: 42
Length: 00 16
Nonce: 00 01 00 00 00 00 00 05
Ciphertext: ae a0 15 56 67 92
ICV: 4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 with Connection ID gives 30 bytes overhead.

2.1.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the sequence number ‘01’ used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.
Compressed DTLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
b0 c3 04 05 42 00 16 f2 0e ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.2 Record Layer Header and Nonce:
b0 c3 04 05 42 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters
(epoch, sequence number, Connection ID, length) gives 17 bytes
overhead.

2.2. DTLS 1.3

2.2.1. DTLS 1.3

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13].
The changes compared to DTLS 1.2 are: omission of version number,
merging of epoch and sequence number fields (of total 8 bytes) into
one 4-bytes-field.

DTLS 1.3 Record Layer (22 bytes, 16 bytes overhead):
17 40 00 00 05 00 0f ae a0 15 56 67 92 ec 4d ff
8a 24 e4 cb 35 b9

Content type:
17
Epoch and Sequence:
40 00 00 05
Length:
00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 16 bytes overhead.

2.2.2. DTLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13]
when compressed with [RFC7400] [OlegHahm-ghc].
Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
02 17 40 80 12 05 00 0f af a0 15 56 67 92 ec 4d
ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
02 17 40 80 12 05 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.2.3. DTLS 1.3 with Connection ID

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id].

DTLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
17 40 00 00 05 42 00 0f ae a0 15 56 67 92 ec 4d
ff 8a 24 e4 cb 35 b9

Content type:
17
Epoch and Sequence:
40 00 00 05
Connection ID:
42
Length:
00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 17 bytes overhead.

2.2.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with [RFC7400] [OlegHahm-ghc].
Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (24 bytes, 18 bytes overhead):
02 17 40 80 13 05 42 00 0f ae a0 15 56 67 92 ec
4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
02 17 40 80 13 05 42 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, Connection ID, length) gives 18 bytes overhead.

2.2.5. DTLS 1.3 with short header

This section analyzes the overhead of DTLS 1.3 with short header format [I-D.ietf-tls-dtls13]. The short header format for DTLS 1.3 reduces the header of 5 bytes, by omitting the length value and sending 1 lower bit of epoch value instead of 2, and 12 lower bits of sequence number instead of 30.

DTLS 1.3 Record Layer (17 bytes, 11 bytes overhead):
30 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 short header:
30 05
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 with short header gives 11 bytes overhead.

2.2.6. DTLS 1.3 with short header and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 with short header [I-D.ietf-tls-dtls13] when compressed with [RFC7400] [OlegHahm-ghc].
Compressed DTLS 1.3 Record Layer (18 bytes, 12 bytes overhead)
11 30 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 short header (including sequence number)
11 30 05
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.3 with short header gives 12 bytes overhead.

2.3. TLS 1.2

2.3.1. TLS 1.2

This section analyzes the overhead of TLS 1.2 [RFC5246]. The changes compared to DTLS 1.2 is that the TLS 1.2 record layer does not have epoch and sequence number, and that the version is different.

TLS 1.2 Record Layer (27 bytes, 21 bytes overhead):
17 03 03 00 16 00 00 00 00 00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 03
Length:
00 16
Nonce:
00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.2 gives 21 bytes overhead.

2.3.2. TLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.2 [RFC5246] when compressed with [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.
Compressed TLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
05 17 03 03 00 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.2 Record Layer Header and Nonce:
05 17 03 03 00 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.2 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.4. TLS 1.3

2.4.1. TLS 1.3

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13]. The change compared to TLS 1.2 is that the TLS 1.3 record layer uses a different version.

TLS 1.3 Record Layer (20 bytes, 14 bytes overhead):
17 03 03 00 16 ae a0 15 56 67 92 ec 4d ff 8a 24
ea4 cb 35 b9

Content type:
17
Legacy Version:
03 03
Length:
00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 14 bytes overhead.

2.4.2. TLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13] when compressed with [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.
Compressed TLS 1.3 Record Layer (21 bytes, 15 bytes overhead)
14 17 03 03 00 0f ae a0 15 56 67 92 ec 4d ff 8a
24 e4 cb 35 b9

Compressed TLS 1.3 Record Layer Header and Nonce:
14 17 03 03 00 0f
Ciphertext (including encrypted ContentType):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.3 with the above parameters
(epoch, sequence number, length) gives 15 bytes overhead.

2.5. OSCORE

This section analyzes the overhead of OSCORE
[I-D.ietf-core-object-security].

Note that Sender ID = ‘’ (empty string) can only be used by one
client per server.

The examples below assume that the original messages does not have
payload (note that this does not affect the overhead).

The below calculation Option Delta = ‘9’, Sender ID = ‘’ (empty
string), and Sequence Number = ‘05’, and is only an example.

OSCORE Request (19 bytes, 13 bytes overhead):
92 09 05
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
92
Option Value (flag byte and sequence number):
09 05
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation Option Delta = ‘9’, Sender ID = ‘42’, and
Sequence Number = ‘05’, and is only an example.
OSCORE Request (20 bytes, 14 bytes overhead):
93 09 05 42
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
93
Option Value (flag byte, sequence number, and Sender ID):
09 05 42
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation uses Option Delta = ’9’.

OSCORE Response (17 bytes, 11 bytes overhead):
90
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Delta and Option Length:
90
Option Value
-
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCORE with the above parameters gives 13-14 bytes overhead for requests and 11 bytes overhead for responses.

Unlike DTLS and TLS, OSCORE has much smaller overhead for responses than requests.

3. Overhead with Different Parameters

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs with the same length.

The compression overhead (GHC) is dependent on the parameters epoch, sequence number, Connection ID, and length (where applicable). The following overheads should be representative for sequence numbers and Connection IDs with the same length.
The OSCORE overhead is dependent on the included CoAP Option numbers as well as the length of the OSCORE parameters Sender ID and sequence number. The following overheads apply for all sequence numbers and Sender IDs with the same length.

<table>
<thead>
<tr>
<th>Sequence Number</th>
<th>05</th>
<th>1005</th>
<th>100005</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS 1.2</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>DTLS 1.3</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>DTLS 1.3 (short header)</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>DTLS 1.2 (GHC)</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>DTLS 1.3 (GHC)</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>DTLS 1.3 (short header) (GCH)</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>TLS 1.2</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>TLS 1.3</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>TLS 1.2 (GHC)</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>TLS 1.3 (GHC)</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>OSCORE Request</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>OSCORE Response</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 1: Overhead in bytes as a function of sequence number (Connection/Sender ID = '')

<table>
<thead>
<tr>
<th>Connection/Sender ID</th>
<th>''</th>
<th>'42'</th>
<th>'4002'</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS 1.2</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>DTLS 1.3</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>DTLS 1.3 (short header)</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>DTLS 1.2 (GHC)</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>DTLS 1.3 (GHC)</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>DTLS 1.3 (short header) (GCH)</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>OSCORE Request</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>OSCORE Response</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Figure 2: Overhead in bytes as a function of Connection/Sender ID (Sequence Number = '05')
### Figure 3: Overhead (excluding ICV) in bytes

(Connection/Sender ID = ‘’, Sequence Number = ‘05’)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Overhead</th>
<th>Overhead (GHC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS 1.2</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>DTLS 1.3</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>DTLS 1.3 (short header)</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>TLS 1.2</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>TLS 1.3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>OSCORE Request</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>OSCORE Response</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

#### 4. Summary

DTLS 1.2 has quite a large overhead as it uses an explicit sequence number and an explicit nonce. TLS 1.2 has significantly less (but not small) overhead. TLS 1.3 and DTLS 1.3 have quite small overhead. OSCORE and DTLS 1.3 with short header format has very small overhead.

The Generic Header Compression (6LoWPAN-GHC) can in addition to DTLS 1.2 handle TLS 1.2, and DTLS 1.2 with Connection ID. The Generic Header Compression (6LoWPAN-GHC) works very well for Connection ID and the overhead seems to increase exactly with the length of the Connection ID (which is optimal). The compression of TLS 1.2 is not as good as the compression of DTLS 1.2 (as the static dictionary only contains the DTLS 1.2 version number). Similar compression levels as for DTLS could be achieved also for TLS 1.2, but this would require different static dictionaries. For TLS 1.3 and DTLS 1.3, GHC increases the overhead. The 6LoWPAN-GHC header compression is not available when (D)TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

The short header format for DTLS 1.3 reduces the header of 5 bytes, by omitting the length value and sending 1 lower bit of epoch value instead of 2, and 12 lower bits of sequence number instead of 30. This may create problems reconstructing the full sequence number, if ~2000 datagrams in sequence are lost.

OSCORE has much lower overhead than DTLS 1.2 and TLS 1.2. The overhead of OSCORE is smaller than DTLS 1.2 and TLS 1.2 over 6LoWPAN with compression, and this small overhead is achieved even on deployments without 6LoWPAN or 6LoWPAN without DTLS compression. OSCORE is lightweight because it makes use of some excellent features in CoAP, CBOR, and COSE.
5. Security Considerations

This document is purely informational.

6. IANA Considerations

This document has no actions for IANA.

7. Informative References

[I-D.ietf-core-object-security]

[I-D.ietf-tls-dtls-connection-id]

[I-D.ietf-tls-dtls13]

[I-D.ietf-tls-tls13]

[OlegHahm-ghc]

[RFC5246]

[RFC6347]
Acknowledgments

The authors want to thank Ari Keraenen, Carsten Bormann, Goeran Selander, and Hannes Tschofenig for comments and suggestions on previous versions of the draft.

All 6LoWPAN-GHC compression was done with [OlegHahm-ghc].

Authors’ Addresses

John Mattsson
Ericsson AB
Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Email: francesca.palombini@ericsson.com
Abstract

This document specifies how to represent Montgomery curves and (twisted) Edwards curves as curves in short-Weierstrass form and illustrates how this can be used to implement elliptic curve computations using existing implementations that already implement, e.g., ECDSA and ECDH using NIST prime curves.

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 RFC 2119 [RFC2119].

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 May 17, 2018.

Copyright Notice

Copyright (c) 2017 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
1. Fostering Code Reuse with New Elliptic Curves

It is well-known that elliptic curves can be represented using different curve models. Recently, IETF has standardized elliptic curves that are claimed to have better performance and improved robustness against "real world" attacks than curves represented in the traditional "short" Weierstrass model. This draft specifies an alternative representation of points of Curve25519, a so-called Montgomery curve, and of points of Edwards25519, a so-called twisted Edwards curve, which are both specified in [RFC7748], as points of a
specific so-called "short" Weierstrass curve, called Wei25519. The draft also defines how to efficiently switch between these different representations.

Use of Wei25519 allows easy definition of signature schemes and key agreement schemes already specified for traditional NIST prime curves, thereby allowing easy integration with existing specifications, such as NIST SP 800-56a [SP-800-56a], FIPS Pub 186-4 [FIPS-186-4], and ANSI X9.62-2005 [ANSI-X9.62] and fostering code reuse on platforms that already implement some of these schemes using elliptic curve arithmetic for curves in "short" Weierstrass form (see Appendix B.1).

2. Specification of Wei25519

For the specification of Wei25519 and its relationship to Curve25519 and Edwards25519, see Appendix D. For further details and background information on elliptic curves, we refer to the other appendices.

3. Example Uses

3.1. ECDSA-SHA256-25519

RFC 8032 [RFC8032] specifies the use of EdDSA, a "full" Schnorr signature scheme, with instantiation by Edwards25519 and Ed448, two so-called twisted Edwards curves. These curves can also be used with the widely implemented signature scheme ECDSA [FIPS-186-4], by instantiating ECDSA with the curve Wei25519 and hash function SHA-256, where "under the hood" an implementation may carry out elliptic curve scalar multiplication routines using the corresponding representations of a point of the curve Wei25519 in Weierstrass form as a point of the Montgomery curve Curve25519 or of the twisted Edwards curve Edwards25519. (The corresponding ECDSA-SHA512-448 scheme arises if one were to specify a curve in short-Weierstrass form corresponding to Ed448 and use the hash function SHA512.) Note that, in either case, one can implement these schemes with the same representation conventions as used with existing NIST specifications, including bit/byte-ordering, compression functions, and the-like. This allows implementations of ECDSA with the hash function SHA-256 and with the NIST curve P-256 or with the curve Wei25519 specified in this draft to use the same implementation (instantiated with, respectively, the NIST P-256 elliptic curve domain parameters or with the domain parameters of curve Wei25519 specified in Appendix D).
3.2. Other Uses

Any existing specification of cryptographic schemes using elliptic curves in Weierstrass form and that allows introduction of a new elliptic curve (here: Wei25519) is amenable to similar constructs, thus spawning "offspring" protocols, simply by instantiating these using the new curve in "short" Weierstrass form, thereby allowing code and/or specifications reuse and, for implementations that so desire, carrying out curve computations "under the hood" on Montgomery curve and twisted Edwards curve cousins hereof (where these exist). This would simply require definition of a new object identifier for any such envisioned "offspring" protocol. This could significantly simplify standardization of schemes and help keeping the resource and maintenance cost of implementations supporting algorithm agility [RFC7696] at bay.

4. Security Considerations

The different representations of elliptic curve points discussed in this draft are all obtained using a publicly known transformation. Since this transformation is an isomorphism, this transformation maps elliptic curve points to equivalent mathematical objects.

5. IANA Considerations

There is *currently* no IANA action required for this document. New object identifiers would be required in case one wishes to specify one or more of the "offspring" protocols exemplified in Section 3.

6. Normative References


Appendix A. Some (non-Binary) Elliptic Curves

A.1. Curves in short-Weierstrass Form

Let GF(q) denote the finite field with q elements, where q is an odd prime power and where q is not divisible by three. Let W_{a,b} be the Weierstrass curve with defining equation \( y^2 = x^3 + a*x + b \), where a and b are elements of GF(q) and where \( 4*a^3 + 27*b^2 \) is nonzero. The points of W_{a,b} are the ordered pairs (x, y) whose coordinates are elements of GF(q) and which satisfy the defining equation (the so-called "affine points"), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix B.1 for details of the group operation.

A.2. Montgomery Curves

Let GF(q) denote the finite field with q elements, where q is an odd prime power. Let M_{A,B} be the Montgomery curve with defining equation \( B*v^2 = u^3 + A*u^2 + u \), where A and B are elements of GF(q) with A unequal to (+/-)2 and with B nonzero. The points of M_{A,B} are the ordered pairs (u, v) whose coordinates are elements of GF(q) and which satisfy the defining equation (the so-called affine points), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the
identity element. See Appendix B.2 for details of the group
operation.

A.3. Twisted Edwards Curves

Let $GF(q)$ denote the finite field with $q$ elements, where $q$ is an odd
prime power. Let $E_{a,d}$ be the twisted Edwards curve with defining
equation $a*x^2 + y^2 = 1 + d*x^2*y^2$, where $a$ and $d$ are distinct
nonzero elements of $GF(q)$. The points of $E_{a,d}$ are the ordered
pairs $(x, y)$ whose coordinates are elements of $GF(q)$ and which
satisfy the defining equation (the so-called affine points). It can
be shown that this set forms a group under addition if $a$ is a square
in $GF(q)$, whereas $d$ is not, where the point $(0, 1)$ serves as the
identity element. (Note that the identity element satisfies the
defining equation.) See Appendix B.3 for details of the group
operation. An Edwards curve is a twisted Edwards curve with $a=1$.

Appendix B. Elliptic Curve Group Operations

B.1. Group Law for Weierstrass Curves

For each point $P$ on the Weierstrass curve $W_{a,b}$, the point at
infinity $O$ serves as identity element, i.e., $P + O = O + P = P$.

For each point $P:=(x, y)$ on the Weierstrass curve $W_{a,b}$, the point
$-P$ is the point $(x, -y)$ and one has $P + (-P) = O$.

Let $P_1:=(x_1, y_1)$ and $P_2:=(x_2, y_2)$ be distinct points on the
Weierstrass curve $W_{a,b}$ and let $Q:=P_1 + P_2$, where $Q$ is not the
identity element. Then $Q:=(x, y)$, where

$$x + x_1 + x_2 = \lambda^2$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

Let $P:=(x_1, y_1)$ be a point on the Weierstrass curve $W_{a,b}$ and let
$Q:=2P$, where $Q$ is not the identity element. Then $Q:=(x, y)$, where

$$x + 2x_1 = \lambda^2$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (3x_1^2 + a)/(2y_1)$.

B.2. Group Law for Montgomery Curves

For each point $P$ on the Montgomery curve $M_{A,B}$, the point at
infinity $O$ serves as identity element, i.e., $P + O = O + P = P$.

For each point $P:=(x, y)$ on the Montgomery curve $M_{A,B}$, the point
$-P$ is the point $(x, -y)$ and one has $P + (-P) = O$. 
Let $P_1 := (x_1, y_1)$ and $P_2 := (x_2, y_2)$ be distinct points on the Montgomery curve $M_{A,B}$ and let $Q := P_1 + P_2$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + x_1 + x_2 = B \lambda^2 - A$$
$$y + y_1 = \lambda (x_1 - x),$$

where

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

Let $P := (x_1, y_1)$ be a point on the Montgomery curve $M_{A,B}$ and let $Q := 2P$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + 2x_1 = B \lambda^2 - A$$
$$y + y_1 = \lambda (x_1 - x),$$

where

$$\lambda = \frac{3x_1^2 + 2A + 1}{2y_1}.$$

Alternative and more efficient group laws exist, e.g., when using the so-called Montgomery ladder. Details are out of scope.

B.3. Group Law for Twisted Edwards Curves

Note: The group laws below hold for twisted Edwards curves $E_{a,d}$ where $a$ is a square in $GF(q)$, whereas $d$ is not. In this case, the addition formulae below are defined for each pair of points, without exceptions. Generalizations of this group law to other twisted Edwards curves are out of scope.

For each point $P$ on the twisted Edwards curve $E_{a,d}$, the point $O := (0, 1)$ serves as identity element, i.e., $P + O = O + P = P$.

For each point $P := (x, y)$ on the twisted Edwards curve $E_{a,d}$, the point $-P$ is the point $(-x, y)$ and one has $P + (-P) = O$.

Let $P_1 := (x_1, y_1)$ and $P_2 := (x_2, y_2)$ be points on the twisted Edwards curve $E_{a,d}$ and let $Q := P_1 + P_2$. Then $Q := (x, y)$, where

$$x = \frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}$$
$$y = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}.$$

Let $P := (x_1, y_1)$ be a point on the twisted Edwards curve $E_{a,d}$ and let $Q := 2P$. Then $Q := (x, y)$, where

$$x = \frac{2x_1y_1}{1 + dx_1^2y_1^2}$$
$$y = \frac{y_1^2 - ax_1^2}{1 - dx_1^2y_1^2}.$$

Note that one can use the formulae for point addition to implement point doubling, taking inverses and adding the identity element as well (i.e., the point addition formulae are uniform and complete (subject to our Note above)).
Appendix C. Relationship Between Curve Models

The non-binary curves specified in Appendix A are expressed in different curve models, viz. as curves in short-Weierstrass form, as Montgomery curves, or as twisted Edwards curves. These curve models are related, as follows.

C.1. Mapping between twisted Edwards Curves and Montgomery Curves

One can map points of the Montgomery curve $M_{A,B}$ to points of the twisted Edwards curve $E_{a,d}$, where $a := (A+2)/B$ and $d := (A-2)/B$ and, conversely, map points of the twisted Edwards curve $E_{a,d}$ to points of the Montgomery curve $M_{A,B}$, where $A := 2(a+d)/(a-d)$ and where $B := 4/(a-d)$. For twisted Edwards curves we consider (i.e., those where $a$ is a square in $GF(q)$, whereas $d$ is not), this defines a one-to-one correspondence, which - in fact - is an isomorphism between $M_{A,B}$ and $E_{a,d}$, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

For the Montgomery curves and twisted Edwards curves we consider, the mapping from $M_{A,B}$ to $E_{a,d}$ is defined by mapping the point at infinity $O$ and the point $(0, 0)$ of order two on $M_{A,B}$ to, respectively, the point $(0, 1)$ and the point $(0, -1)$ of order two of $E_{a,d}$, while mapping each other point $(u, v)$ on $M_{A,B}$ to the point $(x, y) := (u/v, (u-1)/(u+1))$ on $E_{a,d}$. The inverse mapping from $E_{a,d}$ to $M_{A,B}$ is defined by mapping the point $(0, 1)$ and the point $(0, -1)$ of order two of $E_{a,d}$ to, respectively, the point at infinity $O$ and the point $(0, 0)$ of order two on $M_{A,B}$, while each other point $(x, y)$ of $E_{a,d}$ is mapped to the point $(u, v) := ((1+y)/(1-y), (1+y)/(1-y)*x))$ of $M_{A,B}$.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a twisted Edwards curve on the corresponding Montgomery curve, or vice-versa, and translating the result back to the original curve, thereby potentially allowing code reuse.

C.2. Mapping between Montgomery Curves and Weierstrass Curves

One can map points on the Montgomery curve $M_{A,B}$ to points on the Weierstrass curve $W_{a,b}$, where $a := (3-A^2)/(3*B^2)$ and $b := (2*A^3-9*A)/(27*B^3)$. This defines a one-to-one correspondence, which - in fact - is an isomorphism between $M_{A,B}$ and $W_{a,b}$, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

The mapping from $M_{A,B}$ to $W_{a,b}$ is defined by mapping the point at infinity $O$ on $M_{A,B}$ to the point at infinity $O$ on $W_{a,b}$, while
mapping each other point \((u, v)\) of \(M_{A,B}\) to the point \((x, y) := (u/B + A/(3*B), v/B)\) of \(W_{a,b}\). Note that not all Weierstrass curves can be injectively mapped to Montgomery curves, since the latter have a point of order two and the former may not. In particular, if a Weierstrass curve has prime order, such as is the case with the so-called "NIST curves", this inverse mapping is not defined.

This mapping can be used to implement elliptic curve group operations originally defined for a twisted Edwards curve or for a Montgomery curve using group operations on the corresponding elliptic curve in short-Weierstrass form and translating the result back to the original curve, thereby potentially allowing code reuse. Note that implementations for elliptic curves with short-Weierstrass form that hard-code the domain parameter \(a\) to \(a = -3\) (which value is known to allow more efficient implementations) cannot always be used this way, since the curve \(W_{a,b}\) may not always be expressed in terms of a Weierstrass curve with \(a = -3\) via a coordinate transformation.

C.3. Mapping between twisted Edwards Curves and Weierstrass Curves

One can map points of the twisted Edwards curve \(E_{a,d}\) to points of the Weierstrass curve \(W_{a,b}\), via function composition, where one uses the isomorphic mapping between twisted Edwards curve and Montgomery curves of Appendix C.1 and the one between Montgomery and Weierstrass curves of Appendix C.2. Obviously, one can use function composition (now using the respective inverses) to realize the inverse of this mapping.

Appendix D. Curve25519 and Cousins

D.1. Curve Definition and Alternative Representations

The elliptic curve Curve25519 is the Montgomery curve \(M_{A,B}\) defined over the prime field \(GF(p)\), with \(p := 2^{255} - 19\), where \(A := 486662\) and \(B := 1\). This curve has order \(h*n\), where \(h = 8\) and where \(n\) is a prime number. For this curve, \(A^2 - 4\) is not a square in \(GF(p)\), whereas \(A + 2\) is. The quadratic twist of this curve has order \(h1*n1\), where \(h1 = 4\) and where \(n1\) is a prime number. For this curve, the base point is defined to be the ordered pair \((Gu, Gv)\) of elements of \(GF(p)\), where \(Gu = 9\) and where \(Gv\) is an odd integer in the interval \([0, p-1]\).

This curve has the same group structure as (is "isomorphic to") the twisted Edwards curve \(E_{a,d}\) defined over \(GF(p)\), with as base point the ordered pair \((Gx, Gy)\) of elements of \(GF(p)\), where parameters are as specified in Appendix D.3. This curve is denoted as Edwards25519. For this curve, the parameter \(a\) is a square in \(GF(p)\), whereas \(d\) is not, so the group laws of Appendix B.3 apply.
The curve is also isomorphic to the elliptic curve \( W_{(a,b)} \) in short-Weierstrass form defined over \( \text{GF}(p) \), with as base point the ordered pair \((Gx',Gy')\) of elements, where parameters are as specified in Appendix D.3. This curve is denoted as Wei25519.

D.2. Switching between Alternative Representations

Each point \((u,v)\) of Curve25519 corresponds to the point \((x,y) := (u + A/3, y)\) of Wei25519, while the point at infinity of Curve25519 corresponds to the point at infinity of Wei25519. (Here, we used the mapping of Appendix C.2.) Under this mapping, the base point \((Gu,Gv)\) of Curve25519 corresponds to the base point \((Gx',Gy')\) of Wei25519. The inverse mapping maps the point \((x,y)\) on Wei25519 to \((u,v) := (x - A/3, y)\) on Curve25519, while mapping the point at infinity of Wei25519 to the point at infinity on Curve25519. Note that this mapping involves a simple shift of the first coordinate and can be implemented via integer-only arithmetic as a shift of \((p+A)/3\) for the isomorphic mapping and a shift of \(- (p+A)/3\) for its inverse, where \(\delta=(p+A)/3\) is the element of \(\text{GF}(p)\) defined by

\[
\delta = 192986815395526992372618308347813179755449974427342739909597334652188435537
\]

\(=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2451\)

The curve Edwards25519 is isomorphic to the curve Curve25519, where the base point \((Gu,Gv)\) of Curve25519 corresponds to the base point \((Gx,Gy)\) of Edwards25519 and where the point at infinity and the point \((0,0)\) of order two of Curve25519 correspond to, respectively, the point \((0,1)\) and the point \((0,-1)\) of order two of Edwards25519 and where each other point \((u,v)\) of Curve25519 corresponds to the point \((c*u/v, (u-1)/(u+1))\) of Edwards25519, where \(c\) is the element of \(\text{GF}(p)\) defined by

\[
c = \sqrt{-(A+2)}
\]

\[
51042569399160536130206135233146329284152202253034631822681833788666877215207
\]

\(=0x70d9120b 9f5ff944 2d84f723 fc03b081 3a5e2c2e b482e57d 3391fb55 00ba81e7\)

(Here, we used the mapping of Appendix C.1.) The inverse mapping from Edwards25519 to Curve25519 is defined by mapping the point \((0,1)\) and the point \((0,-1)\) of order two of Edwards25519 to, respectively, the point at infinity and the point \((0,0)\) of order two.
of Curve25519 and having each other point \((x, y)\) of Edwards25519 correspond to the point \(((1 + y)/(1 - y), c*(1 + y)/(1-y)*x))\).

The curve Edwards25519 is isomorphic to the Weierstrass curve Wei25519, where the base point \((Gx, Gy)\) of Edwards25519 corresponds to the base point \((Gx', Gy')\) of Wei25519 and where the identity element \((0,1)\) and the point \((0,-1)\) of order two of Edwards25519 correspond to, respectively, the point at infinity \(O\) and the point \((A/3, 0)\) of order two of Wei25519 and where each other point \((x, y)\) of Edwards25519 corresponds to the point \((x', y'):=((1+y)/(1-y)+A/3, c*(1+y)/(1-y)*x))\) of Wei25519, where \(c\) was defined before. \(\text{(Here, we used the mapping of Appendix C.3.)}\) The inverse mapping from Wei25519 to Edwards25519 is defined by mapping the point at infinity \(O\) and the point \((A/3, 0)\) of order two of Wei25519 to, respectively, the identity element \((0,1)\) and the point \((0,-1)\) of order two of Edwards25519 and having each other point \((x, y)\) of Wei25519 correspond to the point \((c*(3*x-A)/(3*y), (3*x-A-3)/(3*x-A+3))\).

Note that these mappings can be easily realized in projective coordinates, using a few field multiplications only, thus allowing switching between alternative representations with negligible relative incremental cost.

D.3. Domain Parameters

The parameters of the Montgomery curve and the corresponding isomorphic curves in twisted Edwards curve and short-Weierstrass form are as indicated below. Here, the domain parameters of the Montgomery curve Curve25519 and of the twisted Edwards curve Edwards25519 are as specified in RFC 7748; the domain parameters of Wei25519 are "new".

General parameters (for all curve models):

\[
p = 2^{255}-19
\]

\[
=0x7fffffffffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
fffffff ffffffffed)
\]

\[
h = 8
\]

\[
n = 72370055773322622139731865630429942408571163593799076060019509382
85454250989
\]

\[
=2^{252} + 0x14def9de a2f79cd6 5812631a 5cf5d3ed)
\]

\[
h1 = 4
\]
Montgomery curve-specific parameters (for Curve25519):
A 486662
B 1
Gu 9 (=0x9)
Gv 14781619447589544791020593568409986887264606134616475288964881837
755586237401
 (=0x20ae19a1 b8a086b4 e01edd2c 7748d14c 923d4d7e 6d7c61b2 29e9c5a2 7eced3d9)

Twisted Edwards curve-specific parameters (for Edwards25519):
a -1 (-0x1)
d -121665/121666
 (=370957059346694393431380835087545651895421138798432190163887855
3308594028355)
 (=0x52036cee 2b6ffe73 8cc74079 7779e898 00700a4d 4141d8ab 75eb4dca 135978a3)
Gx 1511222134953540077250115140958531511454012693041857206046113283
949847762202
 (=0x216936d3 cd6e53fe c0a4e231 fdd6dc5c 692cc760 9525a7b2 c9562d60 8f25d51a)
Gy 4/5
 (=46316835694264781694283940034751631413079938662562256157830336
03165251855960)
 (=0x66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666)

Weierstrass curve-specific parameters (for Wei25519):

Struik  Expires May 17, 2018
a 1929868153955269923726183083478131797554499744273427339909597334
  573241639236
  (=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
     aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa98 4914a144)

b 55751746669818908907645289078257140818241103727901012315294400837
  956729358436
  (=0x7b425ed0 97b425ed 097b425e d097b425 ed097b42 5ed097b4
     260b5e9c 7710c864)

Gx’ 1929868153955269923726183083478131797554499744273427339909597334
   652188435546
   (=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
    aaaaaaaaa aaad245a)

Gy’ 14781619447589544791020593568409986887264606134616475288964881837
   755586237401
   (=0x20ae19a1 b8a086b4 e01edd2c 7748d14c 923d4d7e 6d7c61b2
    29e9c5a2 7eced3d9)

Author’s Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com
Abstract

This document specifies how to represent Montgomery curves and (twisted) Edwards curves as curves in short-Weierstrass form and illustrates how this can be used to implement elliptic curve computations using existing implementations that already implement, e.g., ECDSA and ECDH using NIST prime curves.

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 RFC 2119 [RFC2119].

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 January 20, 2019.

Copyright Notice

Copyright (c) 2018 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
1. Fostering Code Reuse with New Elliptic Curves

It is well-known that elliptic curves can be represented using different curve models. Recently, IETF standardized elliptic curves that are claimed to have better performance and improved robustness against "real world" attacks than curves represented in the traditional "short" Weierstrass model. This draft specifies an alternative representation of points of Curve25519, a so-called Montgomery curve, and of points of Edwards25519, a so-called twisted Edwards curve, which are both specified in [RFC7748], as points of a specific so-called "short" Weierstrass curve, called Wei25519. The draft also defines how to efficiently switch between these different representations.

Use of Wei25519 allows easy definition of signature schemes and key agreement schemes already specified for traditional NIST prime curves, thereby allowing easy integration with existing specifications, such as NIST SP 800-56a [SP-800-56a], FIPS Pub 186-4 [FIPS-186-4], and ANSI X9.62-2005 [ANSI-X9.62] and fostering code reuse on platforms that already implement some of these schemes using elliptic curve arithmetic for curves in "short" Weierstrass form (see Appendix B.1).

2. Specification of Wei25519

For the specification of Wei25519 and its relationship to Curve25519 and Edwards25519, see Appendix D. For further details and background information on elliptic curves, we refer to the other appendices.

The use of Wei25519 allows reuse of existing generic code that implements short-Weierstrass curves, such as the NIST curve P256, to also implement the CFRG curves Curve25519 and Ed25519. The draft also caters to reuse of existing code where some domain parameters may have been hardcoded, thereby widening the scope of applicability; see Appendix F.

3. Example Uses

3.1. ECDSA-SHA256-25519

RFC 8032 [RFC8032] specifies the use of EdDSA, a "full" Schnorr signature scheme, with instantiation by Edwards25519 and Ed448, two so-called twisted Edwards curves. These curves can also be used with the widely implemented signature scheme ECDSA [FIPS-186-4], by instantiating ECDSA with the curve Wei25519 and hash function SHA-256, where "under the hood" an implementation may carry out elliptic curve scalar multiplication routines using the corresponding representations of a point of the curve Wei25519 in Weierstrass form.
as a point of the Montgomery curve Curve25519 or of the twisted Edwards curve Edwards25519. (The corresponding ECDSA-SHA512-448 scheme arises if one were to specify a curve in short-Weierstrass form corresponding to Ed448 and use the hash function SHA512.) Note that, in either case, one can implement these schemes with the same representation conventions as used with existing NIST specifications, including bit/byte-ordering, compression functions, and the-like. This allows implementations of ECDSA with the hash function SHA-256 and with the NIST curve P-256 or with the curve Wei25519 specified in this draft to use the same implementation (instantiated with, respectively, the NIST P-256 elliptic curve domain parameters or with the domain parameters of curve Wei25519 specified in Appendix D).

3.2. Other Uses

Any existing specification of cryptographic schemes using elliptic curves in Weierstrass form and that allows introduction of a new elliptic curve (here: Wei25519) is amenable to similar constructs, thus spawning "offspring" protocols, simply by instantiating these using the new curve in "short" Weierstrass form, thereby allowing code and/or specifications reuse and, for implementations that so desire, carrying out curve computations "under the hood" on Montgomery curve and twisted Edwards curve cousins hereof (where these exist). This would simply require definition of a new object identifier for any such envisioned "offspring" protocol. This could significantly simplify standardization of schemes and help keeping the resource and maintenance cost of implementations supporting algorithm agility [RFC7696] at bay.

4. Security Considerations

The different representations of elliptic curve points discussed in this draft are all obtained using a publicly known transformation. Since this transformation is an isomorphism, this transformation maps elliptic curve points to equivalent mathematical objects.

5. IANA Considerations

There is *currently* no IANA action required for this document. New object identifiers would be required in case one wishes to specify one or more of the "offspring" protocols exemplified in Section 3.

6. Normative References


Appendix A. Some (non-Binary) Elliptic Curves

A.1. Curves in short-Weierstrass Form

Let GF(q) denote the finite field with q elements, where q is an odd prime power and where q is not divisible by three. Let W_{a,b} be the Weierstrass curve with defining equation $y^2 = x^3 + a \cdot x + b$, where a and b are elements of GF(q) and where $4a^3 + 27b^2$ is nonzero. The points of W_{a,b} are the ordered pairs (x, y) whose coordinates are elements of GF(q) and that satisfy the defining equation (the so-called affine points), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix B.1 for details of the group operation.

A.2. Montgomery Curves

Let GF(q) denote the finite field with q elements, where q is an odd prime power. Let M_{A,B} be the Montgomery curve with defining equation $b \cdot v^2 = u^3 + A \cdot u^2 + u$, where A and B are elements of GF(q) with A unequal to (+/-)2 and with B nonzero. The points of M_{A,B} are the ordered pairs (u, v) whose coordinates are elements of GF(q) and that satisfy the defining equation (the so-called affine points), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix B.2 for details of the group operation.

A.3. Twisted Edwards Curves

Let GF(q) denote the finite field with q elements, where q is an odd prime power. Let E_{a,d} be the twisted Edwards curve with defining equation $a \cdot x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2$, where a and d are distinct nonzero elements of GF(q). The points of E_{a,d} are the ordered pairs (x, y) whose coordinates are elements of GF(q) and that satisfy the defining equation (the so-called affine points). It can be shown that this set forms a group under addition if a is a square in GF(q), whereas d is not, where the point (0, 1) serves as the identity element. (Note that the identity element satisfies the defining equation.) See Appendix B.3 for details of the group operation. An Edwards curve is a twisted Edwards curve with a=1.
Appendix B. Elliptic Curve Group Operations

B.1. Group Law for Weierstrass Curves

For each point $P$ of the Weierstrass curve $W_{a,b}$, the point at infinity $O$ serves as identity element, i.e., $P + O = O + P = P$.

For each affine point $P := (x, y)$ of the Weierstrass curve $W_{a,b}$, the point $-P$ is the point $(x, -y)$ and one has $P + (-P) = O$.

Let $P_1 := (x_1, y_1)$ and $P_2 := (x_2, y_2)$ be distinct affine points of the Weierstrass curve $W_{a,b}$ and let $Q := P_1 + P_2$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + x_1 + x_2 = \lambda^2$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

Let $P := (x_1, y_1)$ be an affine point of the Weierstrass curve $W_{a,b}$ and let $Q := 2P$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + 2x_1 = \lambda^2$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (3x_1^2 + a)/(2y_1)$.

Alternative and more efficient group laws exist, e.g., when using the so-called Montgomery ladder. Details are out of scope.

B.2. Group Law for Montgomery Curves

For each point $P$ of the Montgomery curve $M_{A,B}$, the point at infinity $O$ serves as identity element, i.e., $P + O = O + P = P$.

For each affine point $P := (x, y)$ of the Montgomery curve $M_{A,B}$, the point $-P$ is the point $(x, -y)$ and one has $P + (-P) = O$.

Let $P_1 := (x_1, y_1)$ and $P_2 := (x_2, y_2)$ be distinct affine points of the Montgomery curve $M_{A,B}$ and let $Q := P_1 + P_2$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + x_1 + x_2 = B\lambda^2 - A$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

Let $P := (x_1, y_1)$ be an affine point of the Montgomery curve $M_{A,B}$ and let $Q := 2P$, where $Q$ is not the identity element. Then $Q := (x, y)$, where

$$x + 2x_1 = B\lambda^2 - A$$
$$y + y_1 = \lambda(x_1 - x),$$
where $\lambda = (3x_1^2 + 2A + 1)/(2y_1)$. 

Alternative and more efficient group laws exist, e.g., when using the so-called Montgomery ladder. Details are out of scope.
B.3. Group Law for Twisted Edwards Curves

Note: The group laws below hold for twisted Edwards curves $E_{a,d}$ where $a$ is a square in GF(q), whereas $d$ is not. In this case, the addition formulae below are defined for each pair of points, without exceptions. Generalizations of this group law to other twisted Edwards curves are out of scope.

For each point $P$ of the twisted Edwards curve $E_{a,d}$, the point $O=(0,1)$ serves as identity element, i.e., $P + O = O + P = P$.

For each point $P:=(x, y)$ of the twisted Edwards curve $E_{a,d}$, the point $-P$ is the point $(-x, y)$ and one has $P + (-P) = O$.

Let $P_1:=(x_1, y_1)$ and $P_2:=(x_2, y_2)$ be points of the twisted Edwards curve $E_{a,d}$ and let $Q:=P_1 + P_2$. Then $Q:=(x, y)$, where

$$x = \frac{x_1*y_2 + x_2*y_1}{1 + d*x_1*x_2*y_1*y_2} \text{ and } y = \frac{y_1*y_2 - a*x_1*x_2}{1 - d*x_1*x_2*y_1*y_2}.$$

Let $P:=(x_1, y_1)$ be a point of the twisted Edwards curve $E_{a,d}$ and let $Q:=2P$. Then $Q:=(x, y)$, where

$$x = \frac{2*x_1*y_1}{1 + d*x_1^2*y_1^2} \text{ and } y = \frac{y_1^2 - a*x_1^2}{1 - d*x_1^2*y_1^2}.$$

Note that one can use the formulae for point addition to implement point doubling, taking inverses and adding the identity element as well (i.e., the point addition formulae are uniform and complete (subject to our Note above)).

Appendix C. Relationship Between Curve Models

The non-binary curves specified in Appendix A are expressed in different curve models, viz. as curves in short-Weierstrass form, as Montgomery curves, or as twisted Edwards curves. These curve models are related, as follows.

C.1. Mapping between twisted Edwards Curves and Montgomery Curves

One can map points of the Montgomery curve $M_{A,B}$ to points of the twisted Edwards curve $E_{a,d}$, where $a:=(A+2)/B$ and $d:=(A-2)/B$ and, conversely, map points of the twisted Edwards curve $E_{a,d}$ to points of the Montgomery curve $M_{A,B}$, where $A:=2(a+d)/(a-d)$ and $B:=4/(a-d)$. For twisted Edwards curves we consider (i.e., those where $a$ is a square in GF(q), whereas $d$ is not), this defines a one-to-one correspondence, which - in fact - is an isomorphism between
M_{A,B} and E_{a,d}, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

For the Montgomery curves and twisted Edwards curves we consider, the mapping from M_{A,B} to E_{a,d} is defined by mapping the point at infinity O and the point (0, 0) of order two of M_{A,B} to, respectively, the point (0, 1) and the point (0, -1) of order two of E_{a,d}, while mapping each other point (u, v) of M_{A,B} to the point (x, y):=(u/v, (u-1)/(u+1)) of E_{a,d}. The inverse mapping from E_{a,d} to M_{A,B} is defined by mapping the point (0, 1) and the point (0, -1) of order two of E_{a,d} to, respectively, the point at infinity O and the point (0, 0) of order two of M_{A,B}, while each other point (x, y) of E_{a,d} is mapped to the point (u, v):=((1+y)/(1-y), (1+y)/((1-y)*x)) of M_{A,B}.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a twisted Edwards curve on the corresponding Montgomery curve, or vice-versa, and translating the result back to the original curve, thereby potentially allowing code reuse.

C.2. Mapping between Montgomery Curves and Weierstrass Curves

One can map points of the Montgomery curve M_{A,B} to points of the Weierstrass curve W_{a,b}, where a:=(3-A^2)/(3*B^2) and b:=(2*A^3-9*A)/(27*B^3). This defines a one-to-one correspondence, which - in fact - is an isomorphism between M_{A,B} and W_{a,b}, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

The mapping from M_{A,B} to W_{a,b} is defined by mapping the point at infinity O of M_{A,B} to the point at infinity O of W_{a,b}, while mapping each other point (u, v) of M_{A,B} to the point (x, y):=(u/B+A/(3*B), v/B) of W_{a,b}. Note that not all Weierstrass curves can be injectively mapped to Montgomery curves, since the latter have a point of order two and the former may not. In particular, if a Weierstrass curve has prime order, such as is the case with the so-called "NIST curves", this inverse mapping is not defined.

This mapping can be used to implement elliptic curve group operations originally defined for a twisted Edwards curve or for a Montgomery curve using group operations on the corresponding elliptic curve in short-Weierstrass form and translating the result back to the original curve, thereby potentially allowing code reuse. Note that implementations for elliptic curves with short-Weierstrass form that hard-code the domain parameter a to a= -3 (which value is known to allow more efficient implementations) cannot always be used this way,
since the curve \( W_{a,b} \) may not always be expressed in terms of a Weierstrass curve with \( a=-3 \) via a coordinate transformation.

C.3. Mapping between twisted Edwards Curves and Weierstrass Curves

One can map points of the twisted Edwards curve \( E_{a,d} \) to points of the Weierstrass curve \( W_{a,b} \), via function composition, where one uses the isomorphic mapping between twisted Edwards curve and Montgomery curves of Appendix C.1 and the one between Montgomery and Weierstrass curves of Appendix C.2. Obviously, one can use function composition (now using the respective inverses) to realize the inverse of this mapping.

Appendix D. Curve25519 and Cousins

D.1. Curve Definition and Alternative Representations

The elliptic curve Curve25519 is the Montgomery curve \( M_{A,B} \) defined over the prime field \( GF(p) \), with \( p:=2^{255}-19 \), where \( A:=486662 \) and \( B:=1 \). This curve has order \( h*n \), where \( h=8 \) and where \( n \) is a prime number. For this curve, \( A^2-4 \) is not a square in \( GF(p) \), whereas \( A+2 \) is. The quadratic twist of this curve has order \( h1*n1 \), where \( h1=4 \) and where \( n1 \) is a prime number. For this curve, the base point is the point \((Gu,Gv)\), where \( Gu=9 \) and where \( Gv \) is an odd integer in the interval \([0, p-1]\).

This curve has the same group structure as (is "isomorphic" to) the twisted Edwards curve \( E_{a,d} \) defined over \( GF(p) \), with as base point the point \((Gx,Gy)\), where parameters are as specified in Appendix D.3. This curve is denoted as Edwards25519. For this curve, the parameter \( a \) is a square in \( GF(p) \), whereas \( d \) is not, so the group laws of Appendix B.3 apply.

The curve is also isomorphic to the elliptic curve \( W_{a,b} \) in short-Weierstrass form defined over \( GF(p) \), with as base point the point \((Gx',Gy')\), where parameters are as specified in Appendix D.3. This curve is denoted as Wei25519.

D.2. Switching between Alternative Representations

Each affine point \((u,v)\) of Curve25519 corresponds to the point \((x,y):=(u + A/3,y)\) of Wei25519, while the point at infinity of Curve25519 corresponds to the point at infinity of Wei25519. (Here, we used the mapping of Appendix C.2.) Under this mapping, the base point \((Gu,Gv)\) of Curve25519 corresponds to the base point \((Gx',Gy')\) of Wei25519. The inverse mapping maps the affine point \((x,y)\) of Wei25519 to \((u,v):=(x - A/3,y)\) of Curve25519, while mapping the point at infinity of Wei25519 to the point at infinity of Curve25519.
that this mapping involves a simple shift of the first coordinate and can be implemented via integer-only arithmetic as a shift of \((p+\text{A})/3\) for the isomorphic mapping and a shift of \(-(p+\text{A})/3\) for its inverse, where \(\text{delta} = (p+\text{A})/3\) is the element of \(\text{GF}(p)\) defined by

\[
\text{delta} = 192986815395269923726183083478131797554499744273427339909597334652188435537
\]

\[ (=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2451) \]

The curve Edwards25519 is isomorphic to the curve Curve25519, where the base point \((\text{Gu}, \text{Gv})\) of Curve25519 corresponds to the base point \((\text{Gx}, \text{Gy})\) of Edwards25519 and where the point at infinity and the point \((0, 0)\) of order two of Curve25519 correspond to, respectively, the point \((0, 1)\) and the point \((0, -1)\) of order two of Edwards25519 and where each other point \((u, v)\) of Curve25519 corresponds to the point \((c*u/v, (u-1)/(u+1))\) of Edwards25519, where \(c\) is the element of \(\text{GF}(p)\) defined by

\[
c = \sqrt{-(\text{A}+2)}
\]

\[
510425693991605361302061352331463292815220225303463182268183788666877215207
\]

\[ (=0x70d9120b 9f5ff944 2d84f723 fc03b081 3a5e2c2e b482e57d3391fb55 00ba81e7) \]

(Here, we used the mapping of Appendix C.1.) The inverse mapping from Edwards25519 to Curve25519 is defined by mapping the point \((0, 1)\) and the point \((0, -1)\) of order two of Edwards25519 to, respectively, the point at infinity and the point \((0, 0)\) of order two of Curve25519 and having each other point \((x, y)\) of Edwards25519 correspond to the point \(((1 + y)/(1 - y), c*(1 + y)/((1-y)*x))\).

The curve Edwards25519 is isomorphic to the Weierstrass curve Wei25519, where the base point \((\text{Gx}, \text{Gy})\) of Edwards25519 corresponds to the base point \((\text{Gx}', \text{Gy}')\) of Wei25519 and where the identity element \((0,1)\) and the point \((0,-1)\) of order two of Edwards25519 correspond to, respectively, the point at infinity \(O\) and the point \((\text{A}/3, 0)\) of order two of Wei25519 and where each other point \((x, y)\) of Edwards25519 corresponds to the point \((x', y')=((1+y)/(1-y)+\text{A}/3, c*(1+y)/((1-y)*x))\) of Wei25519, where \(c\) was defined before. (Here, we used the mapping of Appendix C.3.) The inverse mapping from Wei25519 to Edwards25519 is defined by mapping the point at infinity \(O\) and the point \((\text{A}/3, 0)\) of order two of Wei25519 to, respectively, the identity element \((0,1)\) and the point \((0,-1)\) of order two of
Edwards25519 and having each other point \((x, y)\) of Wei25519 correspond to the point \((c^3(3x-A)/(3y), (3x-A-3)/(3x-A+3))\).

Note that these mappings can be easily realized in projective coordinates, using a few field multiplications only, thus allowing switching between alternative representations with negligible relative incremental cost.

D.3. Domain Parameters

The parameters of the Montgomery curve and the corresponding isomorphic curves in twisted Edwards curve and short-Weierstrass form are as indicated below. Here, the domain parameters of the Montgomery curve Curve25519 and of the twisted Edwards curve Edwards25519 are as specified in RFC 7748; the domain parameters of Wei25519 are "new".

General parameters (for all curve models):

\[
p = 2^{255} - 19
\]

\[
(=0x7fffffffffffffff ffffffffffffffff ffffffffffffffff ffffffffffffffff ffffffffffffffff ffffffffffffffff ffffffffffffffff)
\]

\[
h = 8
\]

\[
n = 7237005577332262213973186563042994240857116359379907606001950938285454250989
\]

\[
(=2^{252} + 0x14def9de a2f79cd6 5812631a 5cf5d3ed)
\]

\[
h1 = 4
\]

\[
n1 = 144740111154664524427946373126085988481603263447650325797860494125407373907997
\]

\[
(=2^{253} - 0x29bdf3bd 45ef39ac b024c634 b9eba7e3)
\]

Montgomery curve-specific parameters (for Curve25519):

\[
A = 486662
\]

\[
B = 1
\]

\[
Gu = 9 (=0x9)
\]

\[
Gv = 14781619447589544791020593568409986887264606134616475288964881837755586237401
\]
Twisted Edwards curve-specific parameters (for Edwards25519):

\[ a = -1 \quad (-0x01) \]
\[ d = -121665/121666 \]
\[ G_x = 1511221349535400772501151409588531511454012693041857206046113283949847762202 \]
\[ G_y = 4/5 \]

Weierstrass curve-specific parameters (for Wei25519):

\[ a = 1929868153955269923726183083478131797554499744273427339909597334573241639236 \]
\[ b = 5575174669818908907645298078257140818241103727901012315294400837956729358436 \]

Struik  Expires January 20, 2019  [Page 13]
Appendix E. Further Mappings

The non-binary curves specified in Appendix A are expressed in different curve models, viz. as curves in short-Weierstrass form, as Montgomery curves, or as twisted Edwards curves. Within each curve model, further mappings exist that induce a mapping between elliptic curves within each curve model. This can be exploited to force some of the domain parameter to a value that allows a more efficient implementation of the addition formulae.

E.1. Isomorphic Mapping between Weierstrass Curves

One can map points of the Weierstrass curve $W_{a,b}$ to points of the Weierstrass curve $W_{a',b'}$, where $a := a'*u^4$ and $b := b'*u^6$ for some nonzero value $u$ of the finite field $GF(q)$. This defines a one-to-one correspondence, which - in fact - is an isomorphism between $W_{a,b}$ and $W_{a',b'}$, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

The mapping from $W_{a,b}$ to $W_{a',b'}$ is defined by mapping the point at infinity $O$ of $W_{a,b}$ to the point at infinity $O$ of $W_{a',b'}$, while mapping each other point $(x, y)$ of $W_{a,b}$ to the point $(x', y') := (x*u^2, y*u^3)$ of $W_{a',b'}$. The inverse mapping from $W_{a',b'}$ to $W_{a,b}$ is defined by mapping the point at infinity $O$ of $W_{a',b'}$ to the point at infinity $O$ of $W_{a,b}$, while mapping each other point $(x', y')$ of $W_{a',b'}$ to the point $(x, y) := (x/u^2, y/u^3)$ of $W_{a,b}$.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with a generic domain parameter $a$ on a corresponding isomorphic Weierstrass curve with domain parameter $a'$ that has a special form, which is known to allow for more efficient implementations of addition laws, and translating the result back to the original curve. In particular, it is known that such efficiency improvements exist if $a' = -3 \pmod{p}$ and one uses so-called Jacobian coordinates with a particular projective version of the addition laws of Appendix B.1. While not all Weierstrass curves can be put into this form, all traditional NIST curves have domain parameter $a = -3$, while all Brainpool curves [RFC5639] are isomorphic to a Weierstrass curve of this form. For details, we refer to [GECC].
Note that implementations for elliptic curves with short-Weierstrass form that hard-code the domain parameter $a$ to $a'=-3$ (which value is known to allow more efficient implementations) cannot always be used this way, since the curve $\mathbb{W}_{a,b}$ may not always be expressed in terms of a Weierstrass curve with $a'=-3$ via a coordinate transformation: this only holds if $a'/a$ is a fourth power in $\text{GF}(q)$. However, even in this case, one can still express the curve $\mathbb{W}_{a,b}$ in terms of a Weierstrass curve with small $a'$ domain parameter, thereby still allowing a more efficient implementation than with a general $a$ value.

E.2. Isogeneous Mapping between Weierstrass Curves

One can still map points of the Weierstrass curve $\mathbb{W}_{a,b}$ to points of the Weierstrass curve $\mathbb{W}_{a',b'}$, where $a':=-3$ (mod $p$), even if $a'/a$ is not a fourth power in $\text{GF}(q)$. In that case, this mapping cannot be an isomorphism (see Appendix E.1) and, thereby, does not define a one-to-one correspondence. Instead, the mapping is a so-called isogeny (or homomorphism). Since most elliptic curve operations process points of prime order or use so-called "co-factor multiplication", in practice the resulting mapping has similar properties. In particular, one can still take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with domain parameter $a$ unequal to $-3$ (mod $p$) on a corresponding isogenous Weierstrass curve with domain parameter $a'=-3$ (mod $p$) and translating the result back to the original curve. Details of this mapping are outside scope of this document.

Appendix F. Further Cousins of Curve25519

F.1. Further Alternative Representations

The Weierstrass curve Wei25519 is isomorphic to the Weierstrass curve Wei25519.2 defined over GF($p$), with as base point the pair $(G1x,G1y)$, where parameters are as specified in Appendix F.3.

F.2. Further Switching

Each affine point $(x,y)$ of Wei25519 corresponds to the point $(x,y):=(x*u^2,y*u^3)$ of Wei25519.2, where $u$ is the element of GF($p$) defined by

$$u = 477316872488735596725552169064967541950834106999182070293910793636321486119$$

($=0x10e26dacae93602704c7e6cff9efe595764cb5c9e04931f6fdeefc657d4e527$),
while the point at infinity of Wei25519 corresponds to the point at
infinity of Wei25519.2. (Here, we used the mapping of Appendix E.1.)
Under this mapping, the base point \((Gx',Gy')\) of Wei25519 corresponds
to the base point \((G1x',G1y')\) of Wei25519.2. The inverse mapping
maps the affine point \((x,y)\) of Wei25519.2 to \((x,y):=(x/u^2,y/u^3)\) of
Wei25519, while mapping the point at infinity of Wei25519.2 to the
point at infinity of Wei25519. Note that this mapping (and its
inverse) involves a multiplication of both coordinates with fixed
constants \(u^2\) and \(u^3\) (respectively, \(1/u^2\) and \(1/u^3\)), which can be
precomputed.

F.3. Further Domain Parameters

The parameters of the Weierstrass curve with \(a=2\) that is isomorphic
with Wei25519 and the parameters of the Weierstrass curve with \(a=-3\)
that is isogeneous with Wei25519 are as indicated below. Both domain
parameter sets can be exploited directly to derive more efficient
point addition formulae, should an implementation facilitate this.

Weierstrass curve-specific parameters (with \(a=2\)):

\[
\begin{align*}
a & 2 \quad (0x2) \\
\text{b} & \quad 4579340433738339159414415854563976158160282736335993851976016290 \\
& \quad 777777599260 \\
& \quad (0x653e25fa \text{ } 4aa43eb9 \text{ } cc42c61b \text{ } 806bcfd1 \text{ } 0e67bc23 \text{ } 09966e90 \\
& \quad 95a202fe \text{ } 9aac731c) \\
\text{G1x'} & \quad 218726072268944427441327971914352883414836203960572472224621495 \\
& \quad 35754145422686 \\
& \quad (0x305b74fc \text{ } 935f1dad \text{ } d440a88e \text{ } 781f0a81 \text{ } 09d6a68d \text{ } 98c6081a \\
& \quad 660528e2 \text{ } 0746dd5e) \\
\text{G1y'} & \quad 13943617903486429134407723576638679615598775307479919871866321 \\
& \quad 47013341290929 \\
& \quad (0x1ed3cedc \text{ } e78b6b19 \text{ } 5d1c361c \text{ } eld4ef00 \text{ } 5b5b102c \text{ } 99083780 \\
& \quad bf830f7e \text{ } a89021b1) \\
\end{align*}
\]

Weierstrass curve-specific parameters (with \(a=-3\)):

[NOTE: parameters indicated with TBD still to be completed, pending
completion of Sage calculations.]

\[
\begin{align*}
a & \quad -3 \\
\end{align*}
\]
\texttt{(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff

\texttt{ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff)}

\[ b \ [\text{TBD}] \]

\texttt{(=0x[TBD])}

\[ G2x' \ [\text{TBD}] \]

\texttt{(=0x[TBD])}

\[ G2y' \ [\text{TBD}] \]

\texttt{(=0x[TBD])}

Author’s Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com