Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 29, 2018

C. Bormann
Universitaet Bremen TZI

January 25, 2018

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

Table of Contents

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

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, <https://www.rfc-editor.org/info/rfc4944>.

7.2. Informative References

[BOOK] Shelby, Z. and C. Bormann, "6LoWPAN", John Wiley & Sons, Ltd monograph, DOI 10.1002/9780470686218, November 2009.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

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

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: December 31, 2017

C. Gomez
UPC/i2CAT
J. Crowcroft
University of Cambridge
M. Scharf
Nokia

June 29, 2017

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

Table of Contents

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 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.

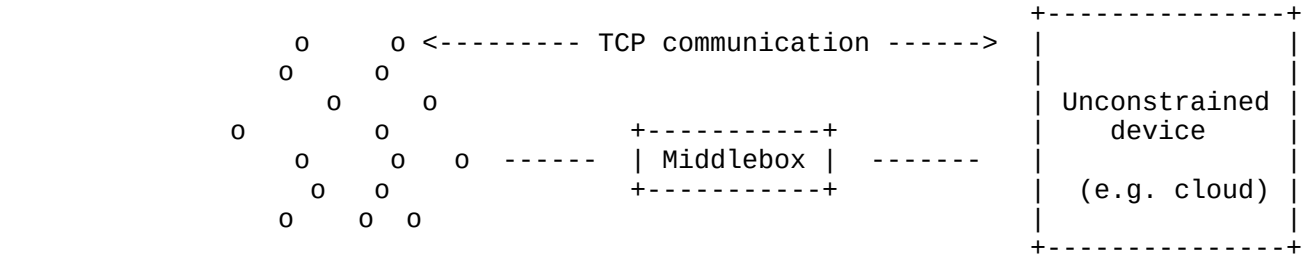

Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

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 receceiving 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, targetted 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 outcoming 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

		uIP	lwIP orig	lwIP 2.0	RIOT	OpenWSN	TinyOS
Memory	Data size	*	*	*	*	*	*
	Code size (kB)	< 5	~9 to ~14	*	*	*	*
TCP features	Window size(MSS)	1	Multiple	Multiple	1	1	*
	Slow start	No	Yes	Yes	No	No	*
	Fast rec/retx	No	Yes	Yes	No	No	*
	Keep-alive	No	*	*	No	No	*
	TFO	No	No	*	No	No	*
	ECN	No	No	*	No	No	*
	Window Scale	No	No	Yes	No	No	*
	TCP timestamps	No	No	Yes	No	No	*
	SACK	No	No	Yes	No	No	*

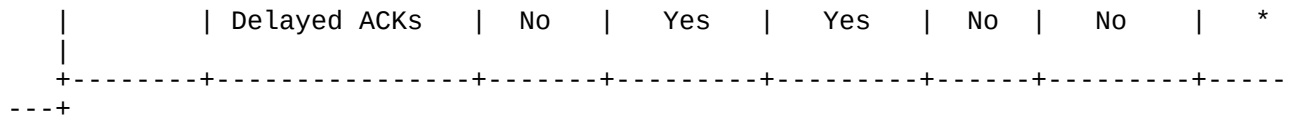

Figure 2: Summary of TCP features for differrent lightweight TCP implementations.

8. References

8.1. Normative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <http://www.rfc-editor.org/info/rfc1323>.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <http://www.rfc-editor.org/info/rfc1981>.

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

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

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <http://www.rfc-editor.org/info/rfc2385>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, DOI 10.17487/RFC2757, January 2000, <http://www.rfc-editor.org/info/rfc2757>.

[RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <http://www.rfc-editor.org/info/rfc2884>.

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

[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, <http://www.rfc-editor.org/info/rfc4944>.

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

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <http://www.rfc-editor.org/info/rfc5925>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <http://www.rfc-editor.org/info/rfc6092>.

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

[RFC6606] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing", RFC 6606, DOI 10.17487/RFC6606, May 2012, <http://www.rfc-editor.org/info/rfc6606>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <http://www.rfc-editor.org/info/rfc7413>.

[RFC7428] Brandt, A. and J. Buron, "Transmission of IPv6 Packets over ITU-T G.9959 Networks", RFC 7428, DOI 10.17487/RFC7428, February 2015, <http://www.rfc-editor.org/info/rfc7428>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <http://www.rfc-editor.org/info/rfc7668>.

[RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <http://www.rfc-editor.org/info/rfc8105>.

[RFC8163] Lynn, K., Ed., Martocci, J., Neilson, C., and S. Donaldson, "Transmission of IPv6 over Master-Slave/Token-Passing (MS/TP) Networks", RFC 8163, DOI 10.17487/RFC8163, May 2017, <http://www.rfc-editor.org/info/rfc8163>.

8.2. Informative References

[Commag] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP Congestion Control for the Internet of Things", IEEE Communications Magazine, June 2016.

[Dunk] A. Dunkels, "Full TCP/IP for 8-Bit Architectures", 2003.

[ETEN] R. Krishnan et al, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks", Computer Networks 2004.

[I-D.delcarpio-6lo-wlanah]
Vega, L., Robles, I., and R. Morabito, "IPv6 over 802.11ah", draft-delcarpio-6lo-wlanah-01 (work in progress), October 2015.

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-01 (work in progress), March 2017.

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-04 (work in progress), June 2017.

[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-07 (work in progress), March 2017.

[I-D.ietf-tcpm-rto-consider]
Allman, M., "Retransmission Timeout Requirements", draft-ietf-tcpm-rto-consider-05 (work in progress), March 2017.

[I-D.tschofenig-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-tschofenig-core-coap-tcp-tls-05 (work in progress), November 2015.

[MQTTS]
U. Hunkeler, H.-L. Truong, A. Stanford-Clark, "MQTT-S: A Publish/Subscribe Protocol For Wireless Sensor Networks", 2008.

Authors' Addresses

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

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

Michael Scharf
Nokia
Lorenzstrasse 10
Stuttgart, 70435
Germany

Email: michael.scharf@nokia.com

LWIG
Internet-Draft
Intended status: Informational
Expires: August 25, 2019

R. Jadhav, Ed.
R. Sahoo
Huawei
S. Duquennoy
Inria
J. Eriksson
Yanzi Networks

February 21, 2019

Neighbor Management Policy for 6LoWPAN
draft-ietf-lwig-nbr-mgmt-policy-03

Abstract

This document describes the problems associated with neighbor cache management in multihop networks involving resource-constrained devices. Thereafter, it also presents a sample neighbor management policy that allows efficient cache management in multihop LLNs (low-power and lossy networks such as LoWPAN) with resource-constrained devices.

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 25, 2019.

Table of Contents

1. Introduction

In a wireless multihop LLN, the node densities (maximum nodes reachable on the same hop) may vary significantly depending upon deployments and scenarios. Examples of such networks is LoWPAN [REF] networks. 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 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 stablized.

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.

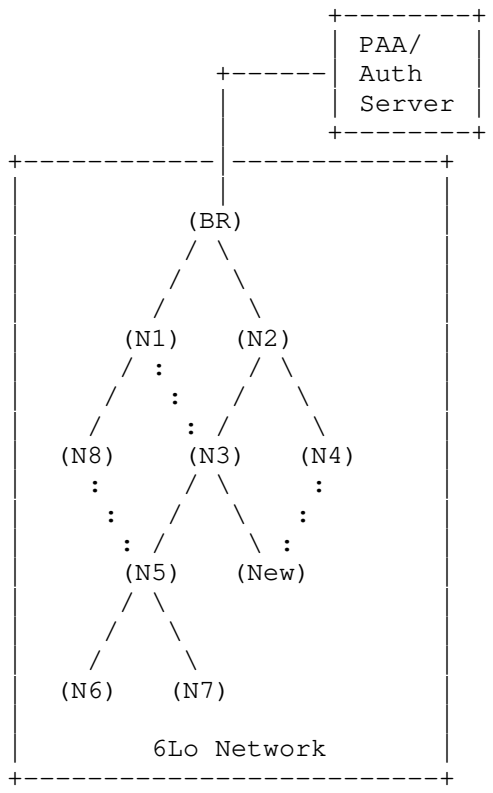

Figure 1: Sample Topology

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].

NDP: Neighbor Discovery protocol [REF]

NS: Neighbor Solicitation

NA: Neighbor Advertisement

LLN: Low Power and Lossy Networks

RPL: Routing Protocol for LLNs [REF]

DAO: DODAG Advertisement Object

DIO: DODAG Information Object

ARO: Address Registration Option defined as part of IPv6 NDP

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.

PCI: PANA Client Initiation is the first message sent by the PaC which initiates the 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 that 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 hueristic 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 roting 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].

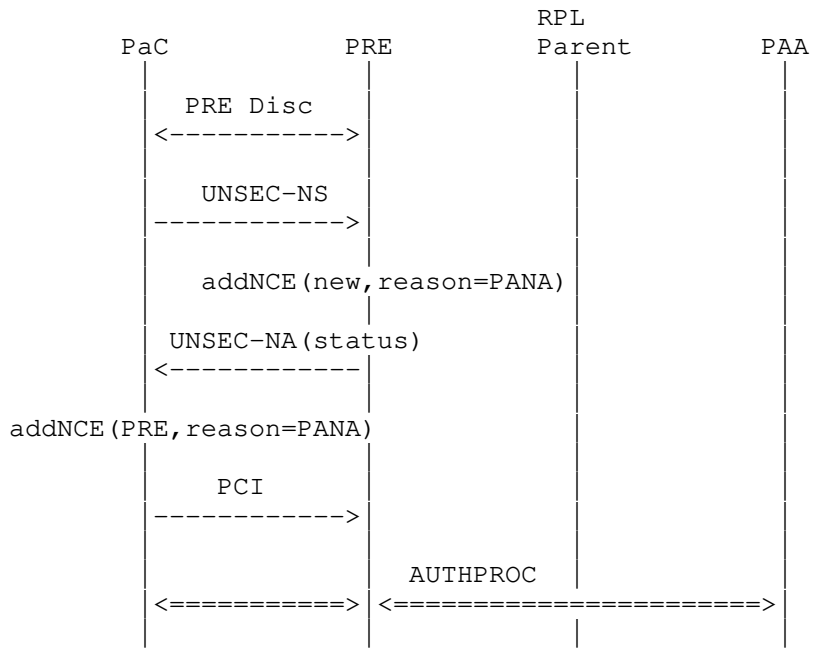

Figure 2: NCE creation between PaC and PRE during relay discovery process

Relay element does not hold any state information on behalf of the new joinee 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.

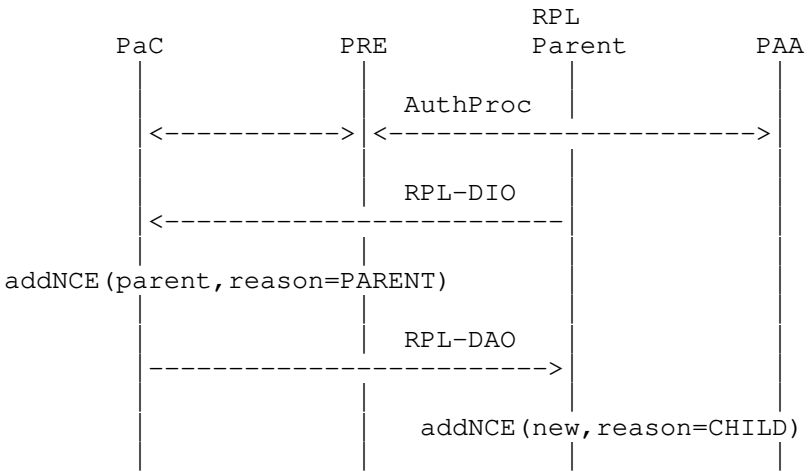

Figure 3: NCE creation call Flow for RPL storing MOP

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.

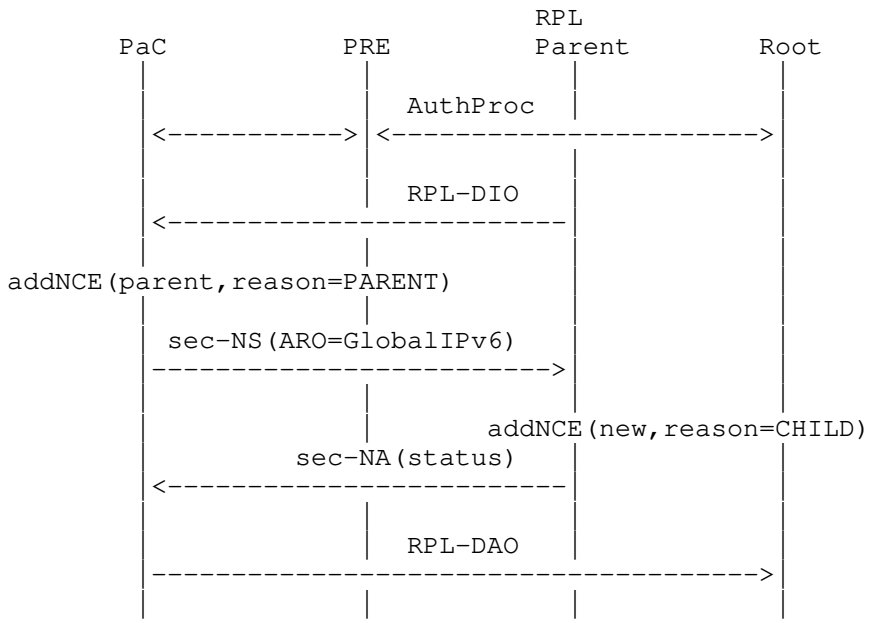

Figure 4: NCE creation call Flow for non-storing MOP

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.

Efficient neighbor cache management depends upon efficient route invalidation since the neighbor cache entries are associated with routing entries. With regards to RPL, issues with the route invalidation has been highlighted in [I-D.ietf-roll-efficient-npdao]. [I-D.ietf-roll-efficient-npdao] also defines a new mechanism for improved route invalidation in storing MOP which helps optimized cleanup of neighbor cache entries.

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 containings 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.

[I-D.richardson-6tisch-roll-enrollment-priority] defines a signalling change in DIO messages to inform child nodes of the priority of the 6LR. The priority field signals whether the 6LR is ready and has enough resources to serve new child nodes. If 6LR's neighbor cache

gets full it can set the min priority to 0x7f(127) to stop the joining process via it. When a LR or leaf node receives DIO from a parent LR with min priority set to 127 the below actions

1. If its not yet joined the DODAG don't choose this LR as preferred parent to join the DODAG.

2. If the node is already in the DODAG and this LR is not in the parent list don't add it to parent list.

3. If node is already in the DODAG and the LR is in its standby parent list remove the LR from its standby parent list.

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:

NCE Entry for	MAX count	Reason
Routing Parent	MAX_ROUTING_PARENT_NCE_NUM	PARENT
Routing child	MAX_ROUTING_CHILD_NCE_NUM	CHILD
Others such as pre-auth sessions	MAX_OTHER_NCE_NUM	OTHER

Table 1: Neighbor Cache Entry reservation

Table 1 denotes that the NCEs are reserved dependending upon the reason for its addition. MAX_ROUTING_PARENT_NCE_NUM specifies maximum number of parent entries a node should allow. MAX_ROUTING_CHILD_NCE_NUM specifies maximum number of child entries that are allowed after which the node SHOULD decline the DAO signalling. MAX_OTHER_NCE_NUM specifies any other entries that might be created. Pre-auth session entries are usually short-lived and should be considered part of this category.

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 unsecured NS/NA. In case of storing MOP, CHILD 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.

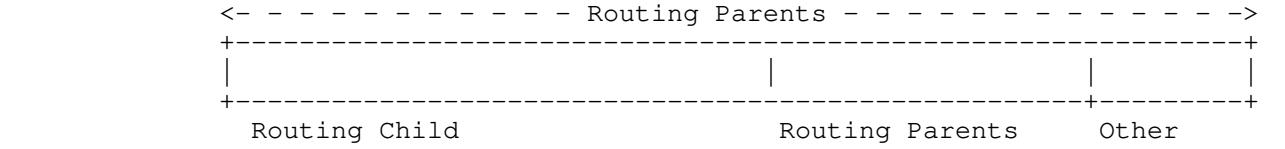

Figure 5: Reservation of NCEs in neighbor table

As shown in the Figure 5, 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 unreacable 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 Mohit Sethi for the review and feedback. 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 to 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:

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

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

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

7.2. Informative References

[CONTIKI] Thingsquare, "Contiki: The Open Source OS for IoT", 2012, <http://www.contiki-os.org>.

[Dawans_et_al] Dawans, S., Duquennoy, S., and O. Bonaventure, "On Link Estimation in Dense RPL Deployments", 2012.

[I-D.ietf-6tisch-architecture] Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-19 (work in progress), December 2018.

[I-D.ietf-6tisch-minimal-security] Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-09 (work in progress), November 2018.

[I-D.ietf-roll-efficient-npdao] Jadhav, R., Thubert, P., Sahoo, R., and Z. Cao, "Efficient Route Invalidation", draft-ietf-roll-efficient-npdao-09 (work in progress), October 2018.

[I-D.richardson-6tisch-roll-enrollment-priority] Richardson, M., "Enabling secure network enrollment in RPL networks", draft-richardson-6tisch-roll-enrollment-priority-02 (work in progress), February 2019.

[LWIP] "lwIP: A Lightweight TCP/IP stack", <https://savannah.nongnu.org/projects/lwip/>.

[RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <https://www.rfc-editor.org/info/rfc5191>.

[RFC6345] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., Ed., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, DOI 10.17487/RFC6345, August 2011, <https://www.rfc-editor.org/info/rfc6345>.

[WHITEFIELD] "Whitefield Framework", <https://github.com/whitefield-framework/whitefield>.

[Woo_et_al]
Woo, A., Tong, T., and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks", 2003.

Appendix A. Performance Result

This appendix provides the details of the performance evaluation of this draft.

Setup: To perform the test [WHITEFIELD] framework was used with LwIP [LWIP] as the network stack. A version of this draft is implemented in the lwip to provide efficient neighbor cache management policy that will work with relatively lower neighbor cache size. RPL is used as routing protocol to form mesh network. The available neighbor table size was split 60%, 30% and 10% among direct child entries, parent entries and others respectively.

Topology: Test was performed with a 64 nodes network including the border router. Grid topology based network was used and all the nodes were in close range with each other simulating a dense condition. 802.15.4 in 2.4GHz range with single channel and un-slotted CSMA wireless RF was used.

Steps: Experiment has been conducted with different neighbor cache sizes 10, 20 and 40. For each NC size we have collected sample readings for packet delivery rate by enabling and disabling the new neighbor Cache Management Policy.

Data transmission frequency: Each node in the network sends 104 bytes (IPv6 Header + RPI + UDP + Data) of UDP request to BR at each 10 second interval. udp Server running at BR process these requests and sends the response back , which is also of same size 104 bytes. A duration of 2 minutes delay is added, for network to get stable, before nodes starts sending request messages at 10 sec interval.To calculate PDR one request and response pair is considered as one successful transaction.

Packet Delivery Rate Performance

Neighbor Cache Size	PDR With New policy	PDR Without New policy
10	96.3	7.8
20	97.5	31.3
40	98.7	98.6

Table 2: Packet delivery rate

Authors' Addresses

Rahul Arvind Jadhav (editor)
Huawei
Kundalahalli Village, Whitefield,
Bangalore, Karnataka 560037
India

Phone: +91-080-49160700
Email: rahul.ietf@gmail.com

Rabi Narayan Sahoo
Huawei
Kundalahalli Village, Whitefield,
Bangalore, Karnataka 560037
India

Phone: +91-080-49160700
Email: rabinarayans@huawei.com

Simon Duquennoy
Inria
40 Avenue Halley
Building A
Villeneuve d'Ascq
France

Phone: +33 768227731
Email: simon.duquennoy@inria.fr

Joakim Eriksson
Yanzi Networks

Email: joakime@sics.se

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2021

C. Gomez
UPC
J. Crowcroft
University of Cambridge
M. Scharf
Hochschule Esslingen
October 30, 2020

TCP Usage Guidance in the Internet of Things (IoT)
draft-ietf-lwig-tcp-constrained-node-networks-13

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 May 3, 2021.

Table of Contents

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. [RFC8352]). New IETF protocol stack components include the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) adaptation layer [RFC4944][RFC6282][RFC6775], the IPv6 Routing Protocol for Low-power and lossy networks (RPL) routing protocol [RFC6550], and the Constrained Application Protocol (CoAP) [RFC7252].

As of the writing, the main current transport layer protocols in IP-based IoT scenarios are UDP and TCP. TCP has been criticized, often unfairly, as a protocol that is unsuitable for the IoT. It is true that some TCP features, such as relatively long header size, unsuitability for multicast, and always-confirmed data delivery, are not optimal for IoT scenarios. 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 is 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 [RFC7230] [RFC7540], 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 Queuing Telemetry Transport (MQTT) [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 code size and the memory requirements. Many TCP extensions are not required for interoperability with other standard-compliant TCP endpoints. Given the limited resources on constrained devices, careful selection of optional TCP features can make an implementation more lightweight.

This document provides guidance on how to implement and configure TCP, as well as on how TCP is advisable to be used by applications, 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. Characteristics of CNNs relevant for TCP

2.1. 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 [RFC8352], 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].

2.2. 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. Alternatively, CNNs may also follow the multihop topology [RFC6606].

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 are also shared for other protocols.

One key use case for TCP in CNNs 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 the first case, communication possibly has to traverse a middlebox (e.g. a firewall, NAT, etc.). Figure 1 illustrates such a 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.

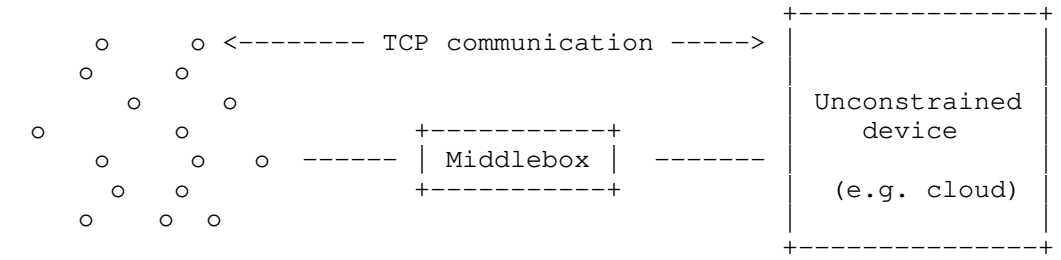

Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

2.3. 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. There is not always 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.

3. 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.

3.1. Addressing path properties

3.1.1. Maximum Segment Size (MSS)

Assuming that IPv6 is used, and 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 IP datagram size to 1280 bytes in order to avoid the need to support Path MTU Discovery [RFC8201]. In addition, an IP datagram size of 1280 bytes avoids incurring IPv6-layer fragmentation [RFC8900].

An IPv6 datagram size exceeding 1280 bytes can be avoided by setting the TCP MSS not larger than 1220 bytes. Note that it is already a requirement that TCP implementations consume payload space instead of increasing datagram size when including IP or TCP options in an IP packet to be sent [RFC6691]. Therefore, it is not required to advertise an MSS smaller than 1220 bytes in order to accommodate TCP options.

Note that setting the MTU to 1280 bytes is possible for link layer technologies in the CNN space, even if some of them 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 [RFC8200], 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) [RFC8376] 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.

Using larger MSS (to a suitable extent) may be beneficial in some scenarios, especially when transferring large payloads, as it reduces the number of packets (and packet headers) required for a given

payload. However, the characteristics of the constrained network need to be considered. In particular, in a lossy network where unreliable fragment delivery is used, the amount of data that TCP unnecessarily retransmits due to fragment loss increases (and throughput decreases) quickly with the MSS. This happens because the loss of a fragment leads to the loss of the whole fragmented packet being transmitted. Unnecessary data retransmission is particularly harmful in CNNs due to the resource constraints of such environments. Note that, while the original 6LoWPAN fragmentation mechanism [RFC4944] does not offer reliable fragment delivery, fragment recovery functionality for 6LoWPAN or 6Lo environments is being standardized as of the writing [I-D.ietf-6lo-fragment-recovery].

3.1.2. Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) [RFC3168] 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.

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. In the context of CNNs, a remarkable feature of ECN is that congestion can be signalled without incurring packet drops (which will lead to retransmissions and consumption of limited resources such as energy and bandwidth).

ECN can further reduce packet losses since congestion control measures can be applied earlier [RFC2884]. Fewer 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 and often impossible 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. In addition, if the constrained device uses power saving techniques, a retransmission timeout will incur a wake-up action, in contrast to ACK clock- triggered sending. When the congestion window of a TCP sender has a size of one segment and a TCP ACK with an ECN signal (ECE flag) arrives at the TCP sender, the TCP sender resets the retransmit timer, and the sender will only be able to send a new packet when the retransmit timer expires. Effectively, the TCP sender reduces at that moment its sending rate from 1 segment per

Round Trip Time (RTT) to 1 segment per Retransmission Timeout (RTO) and reduces the sending rate further on each ECN signal received in subsequent TCP ACKs. Otherwise, if an ECN signal is not present in a subsequent TCP ACK the TCP sender resumes the normal ACK-clocked transmission of segments [RFC3168].

ECN can be incrementally deployed in the Internet. Guidance on configuration and usage of ECN is provided in [RFC7567]. 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. As of the writing, there is on-going work to extend the types of TCP packets that are ECN-capable, including pure ACKs [I-D.ietf-tcpm-generalized-ecn]. Such a feature may further increase the benefits of ECN in CNN environments. Note, however, that supporting ECN increases implementation complexity.

3.1.3. 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 remain as experimental work. In fact, as of today, the IETF has not standardized any such solution.

3.2. TCP guidance for single-MSS stacks

This section discusses TCP stacks that allow transferring a single MSS. More general guidance is provided in Section 3.3.

3.2.1. Single-MSS stacks - benefits and issues

A TCP stack can reduce the memory 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 are 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 length of one MSS, e.g., a firmware download. Furthermore, a single-MSS solution relies solely on timer-based loss recovery, therefore missing the performance gain of Fast Retransmit and Fast Recovery (which require a larger window size, see Section 3.3.1).

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 data unit. For this use of CoAP, a maximum TCP window of one MSS may be sufficient, as long as the CoAP message size does not exceed one MSS. An exception in CoAP over TCP, though, is the Capabilities and Settings Message (CSM) that must be sent at the start of the TCP connection. The first application message carrying user data is allowed to be sent immediately after the CSM message. If the sum of the CSM size plus the application message size exceeds the MSS, a sender using a single-MSS stack will need to wait for the ACK confirming the CSM before sending the application message.

3.2.2. TCP options for single-MSS stacks

A TCP implementation needs to support, at a minimum, TCP options 2, 1 and 0. These are, respectively, the Maximum Segment Size (MSS) option, the No-Operation option, and the End Of Option List marker [RFC0793]. None of these are a substantial burden to support. 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 is permitted to silently ignore all other TCP options.

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 [RFC7323], TCP Timestamps [RFC7323], Selective Acknowledgments (SACK) and SACK-Permitted [RFC2018]. Also other TCP options may not be required on a constrained device with a very lightweight implementation. With regard to the Window scale option, note that it is only useful if a window size greater than 64 kB is needed.

Note that a TCP sender can benefit from the TCP Timestamps option [RFC7323] in detecting spurious RTOs. The latter are quite likely to occur in CNN scenarios due to a number of reasons (e.g. route changes in a multihop scenario, link layer retries, etc.). The header overhead incurred by the Timestamps option (of up to 12 bytes) needs to be taken into account.

3.2.3. Delayed Acknowledgments for single-MSS stacks

TCP Delayed Acknowledgments are meant to reduce the number of ACKs sent within a TCP connection, thus reducing network overhead, but they may increase the time until a sender may receive an ACK. In general, usefulness of Delayed ACKs depends heavily on the usage scenario (see Section 3.3.2). There can be interactions with single-MSS stacks.

When traffic is unidirectional, if the sender can send at most one MSS of data or the receiver advertises a receive window not greater than the MSS, Delayed ACKs may unnecessarily contribute delay (up to 500 ms) to the RTT [RFC5681], which limits the throughput and can increase data delivery time. Note that, in some cases, it may not be possible to disable Delayed ACKs. One known workaround is to split the data to be sent into two segments of smaller size. A standard compliant TCP receiver may immediately acknowledge the second MSS of data, which can improve throughput. However, this 'split hack' may not always work since a TCP receiver is required to acknowledge every second full-sized segment, but not two consecutive small segments. The overhead of sending two IP packets instead of one is another downside of the 'split hack'.

Similar issues may happen when the sender uses the Nagle algorithm, since the sender may need to wait for an unnecessarily delayed ACK to send a new segment. Disabling the algorithm will not have impact if the sender can only handle stop-and-wait operation at the TCP level.

For request-response traffic, when the receiver uses Delayed ACKs, a response to a data message can piggyback an ACK, as long as the latter is sent before the Delayed ACK timer expires, thus avoiding unnecessary ACKs without payload. Disabling Delayed ACKs at the request sender allows an immediate ACK for the data segment carrying the response.

3.2.4. RTO calculation for single-MSS stacks

The RTO calculation is one of the fundamental TCP algorithms [RFC6298]. 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.

If a TCP sender uses a very small window size, and it cannot benefit from Fast Retransmit/Fast Recovery or SACK, the RTO algorithm has a large impact on performance. In that case, RTO algorithm tuning may be considered, although careful assessment of possible drawbacks is recommended [I-D.ietf-tcpm-rto-consider].

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

3.3. 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].

3.3.1. Loss recovery and congestion/flow control

Devices that have enough memory to allow a larger (i.e. more than 3 MSS of data) TCP window size can leverage a more efficient loss recovery than the timer-based approach used for smaller TCP window size (see Section 3.2.1) by using Fast Retransmit and Fast Recovery [RFC5681], at the expense of slightly greater complexity and Transmission Control Block (TCB) size. Assuming that Delayed ACKs are used by the receiver, a window size of up to 5 MSS is required for Fast Retransmit and Fast Recovery to work efficiently: If in a given TCP transmission of full-sized segments 1, 2, 3, 4, and 5, segment 2 gets lost, and the ACK for segment 1 is held by the Delayed ACK timer, then the sender should get an ACK for segment 1 when 3 arrives and duplicate ACKs when segments 4, 5, and 6 arrive. It will retransmit segment 2 when the third duplicate ACK arrives. In order to have segments 2, 3, 4, 5, and 6 sent, the window has to be of at least 5 MSS. With an MSS of 1220 bytes, a buffer of a size of 5 MSS would require 6100 bytes.

The example in the previous paragraph did not use a further TCP improvement such as Limited Transmit [RFC3042]. The latter may also be useful for any transfer that has more than one segment in flight. Small transfers tend to benefit more from Limited Transmit, because they are more likely to not receive enough duplicate ACKs. Assuming the example in the previous paragraph, Limited Transmit allows sending 5 MSS with a congestion window (cwnd) of 3 segments, plus two additional segments for the first two duplicate ACKs. With Limited Transmit, even a cwnd of 2 segments allows sending 5 MSS, at the expense of additional delay contributed by the Delayed ACK timer for the ACK that confirms segment 1.

When a multiple-segment window is used, the receiver will need to manage the reception of possible out-of-order received segments, requiring sufficient buffer space. Note that even when a 1-MSS window is used, out-of-order arrival should also be managed, as the sender may send multiple sub-MSS packets that fit in the window. (On the other hand, the receiver is free to simply drop out-of-order segments, thus forcing retransmissions).

3.3.1.1. Selective Acknowledgments (SACK)

If a device with less severe memory and processing constraints can afford advertising a TCP window size of several MSS, 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. In addition, SACK often allows for faster loss recovery when there is more than one lost segment in a window of data, since SACK recovery may complete with less RTTs. SACK is particularly useful for bulk data transfers. A receiver supporting SACK will need to keep track of the data blocks that need to be received. The sender will also need to keep track of which data segments need to be resent after learning which data blocks are missing at the receiver. 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. When the sender discovers the data blocks that have already been received, it needs to also store the necessary state to avoid unnecessary retransmission of data segments that have already been received.

3.3.2. Delayed Acknowledgments

For certain traffic patterns, Delayed ACKs may have a detrimental effect, as already noted in Section 3.2.3. Advanced TCP stacks may use heuristics to determine the maximum delay for an ACK. For CNNs, the recommendation depends on the expected communication patterns.

When traffic over a CNN is expected to mostly be unidirectional messages with a size typically up to one MSS, and the time between two consecutive message transmissions is greater than the Delayed ACK timeout, it may make sense to use a smaller timeout or disable Delayed ACKs at the receiver. This avoids incurring additional delay, as well as the energy consumption of the sender (which might e.g. keep its radio interface in receive mode) during that time. Note that disabling Delayed ACKs may only be possible if the peer device is administered by the same entity managing the constrained device. For request-response traffic, enabling Delayed ACKs is recommended at the server end, in order to allow combining a response with the ACK into a single segment, thus increasing efficiency. In addition, if a client issues requests infrequently, disabling Delayed ACKs at the client allows an immediate ACK for the data segment carrying the response.

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.

Note that, in many scenarios, the peer that a constrained device communicates with will be a general purpose system that communicates with both constrained and unconstrained devices. Since delayed ACKs are often configured through system-wide parameters, delayed ACKs behavior at the peer will be the same regardless of the nature of the endpoints it talks to. Such a peer will typically have delayed ACKs enabled.

3.3.3. Initial Window

RFC 5681 specifies a TCP Initial Window (IW) of roughly 4 kB [RFC5681]. Subsequently, RFC 6928 defined an experimental new value for the IW, which in practice will result in an IW of 10 MSS [RFC6928]. The latter is nowadays used in many TCP implementations.

Note that a 10-MSS IW was recommended for resource-rich environments (e.g. broadband environments), which are significantly different from CNNs. In CNNs, many application layer data units are relatively small (e.g. below one MSS). However, larger objects (e.g. large files containing sensor readings, firmware updates, etc.) may also need to be transferred in CNNs. If such a large object is transferred in CNNs, with an IW setting of 10 MSS, there is significant buffer overflow risk, since many CNN devices support network or radio buffers of a size smaller than 10 MSS. In order to avoid such problem, in CNNs the IW needs to be carefully set, based on device and network resource constraints. In many cases, a safe IW setting will be smaller than 10 MSS.

4. TCP usage recommendations in CNNs

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

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. Number of concurrent connections

TCP endpoints with a small amount of memory may only support a small number of connections. Each TCP connection requires storing a number of variables in the TCB. Depending on the internal TCP implementation, each connection may result in further memory overhead, and connections may compete for scarce resources (e.g. further memory overhead for send and receive buffers, etc).

A careful application design may try to keep the number of concurrent connections as small as possible. A client can for instance limit the number of simultaneous open connections that it maintains to a given server. Multiple connections could for instance be used to avoid the "head-of-line blocking" problem in an application transfer. However, in addition to consuming resources, using multiple connections can also cause undesirable side effects in congested networks. For example, the HTTP/1.1 specification encourages clients to be conservative when opening multiple connections [RFC7230]. Furthermore, each new connection will start with a 3-way handshake, therefore increasing message overhead.

Being conservative when opening multiple TCP connections is of particular importance in Constrained-Node Networks.

4.3. TCP connection lifetime

In order to minimize message overhead, it makes sense to keep a TCP connection open as long as the two TCP endpoints have more data to send. If applications exchange data rather infrequently, i.e., if TCP connections would stay idle for a long time, the idle time can result in problems. For instance, certain middleboxes such as firewalls or NAT devices are known to delete state records after an inactivity interval. RFC 5382 specifies a minimum value for such interval of 124 minutes. Measurement studies have reported that TCP NAT binding timeouts are highly variable across devices, with a median around 60 minutes, the shortest timeout being around 2 minutes, and more than 50% of the devices with a timeout shorter than the aforementioned minimum timeout of 124 minutes [HomeGateway]. The timeout duration used by a middlebox implementation may not be known to the TCP endpoints.

In CNNs, such middleboxes may e.g. be present at the boundary between the CNN and other networks. If the middlebox can be optimized for CNN use cases, it makes sense to increase the initial value for filter state inactivity timers to avoid problems with idle connections. Apart from that, this problem can be dealt with by different connection handling strategies, each having pros and cons.

One approach for infrequent data transfer is to use short-lived TCP connections. Instead of trying to maintain a TCP connection for a long time, possibly short-lived connections can be opened between two endpoints, which are closed if no more data needs to be exchanged. For use cases that can cope with the additional messages and the latency resulting from starting new connections, it is recommended to use a sequence of short-lived connections, instead of maintaining a single long-lived connection.

The message and latency overhead that stems from using a sequence of short-lived connections could be reduced by TCP Fast Open (TFO) [RFC7413], which is an experimental TCP extension, at the expense of increased implementation complexity and increased TCP Control Block (TCB) size. TFO allows data to be carried in SYN (and SYN-ACK) segments, and to be consumed immediately by the receiving endpoint. This reduces the message and latency overhead compared to the traditional three-way handshake to establish a TCP connection. For security reasons, the connection initiator has to request a TFO 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. While a given cookie is used for multiple connections between the same two endpoints, the latter may become vulnerable to privacy threats. In addition, a valid cookie may be stolen from a compromised host and may be used to perform SYN flood attacks, as well as amplified reflection attacks to victim hosts (see Section 5 of RFC 7413). Nevertheless, TFO is more efficient than frequently opening new TCP connections with the traditional three-way handshake, as long as the cookie can be reused in subsequent connections. However, as stated in RFC 7413, TFO deviates from the standard TCP semantics, since the data in the SYN could be replayed to an application in some rare circumstances. Applications should not use TFO unless they can tolerate this issue, e.g., by using Transport Layer Security (TLS) [RFC7413]. A comprehensive discussion on TFO can be found at RFC 7413.

Another approach is to use long-lived TCP connections with application-layer heartbeat messages. Various application protocols support such heartbeat messages (e.g. CoAP over TCP [RFC8323]). Periodic application-layer heartbeats can prevent early filter state record deletion in middleboxes. If the TCP binding timeout for a middlebox to be traversed by a given connection is known, middlebox filter state deletion will be avoided if the heartbeat period is lower than the middlebox TCP binding timeout. Otherwise, the implementer needs to take into account that middlebox TCP binding timeouts fall in a wide range of possible values [HomeGateway], and it may be hard to find a proper heartbeat period for application-layer heartbeat messages.

One specific advantage of Heartbeat messages is that they also allow aliveness checks at the application level. In general, it makes sense to realize aliveness checks at the highest protocol layer possible that is meaningful to the application, in order to maximize the depth of the aliveness check. In addition, timely detection of a dead peer may allow savings in terms of TCB memory use. However, the transmission of heartbeat messages consumes resources. This aspect needs to be assessed carefully, considering the characteristics of each specific CNN.

A TCP implementation may also be able to send "keep-alive" segments to test a TCP connection. According to [RFC1122], "keep-alives" are an optional TCP mechanism that is turned off by default, i.e., an application must explicitly enable it for a TCP connection. The interval between "keep-alive" messages must be configurable and it must default to no less than two hours. With this large timeout, TCP keep-alive messages might not always be useful to avoid deletion of filter state records in some middleboxes. However, sending TCP keep-alive probes more frequently risks draining power on energy-constrained devices.

5. Security Considerations

Best current practice for securing TCP and TCP-based communication also applies to CNN. As example, use of Transport Layer Security (TLS) [RFC8446] is strongly recommended if it is applicable. However, note that TLS protects only the contents of the data segments.

There are TCP options which can actually protect the transport layer. One example is the TCP Authentication Option (TCP-AO) [RFC5925]. However, this option adds overhead and complexity. TCP-AO typically has a size of 16-20 bytes. An implementer needs to asses the trade-off between security and performance when using TCP-AO, considering the characteristics (in terms of energy, bandwidth and computational power) of the environment where TCP will be used.

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 implementation. 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.

6. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grants CAS15/00336 and and CAS18/00170, and by European Regional Development Fund (ERDF) and the Spanish Government through projects TEC2016-79988-P, PID2019-106808RA-I00, AEI/FEDER, UE, and by Generalitat de Catalunya Grant 2017 SGR 376. Part of his contribution to this work has been carried out during his stays 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, Hannes Tschofenig, David Black, Yoshifumi Nishida, Ilpo Jarvinen, Emmanuel Baccelli, Stuart Cheshire, Gorry Fairhurst, Ingemar Johansson, Ted Lemon, and Michael Tuexen. 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 kindly performed code size measurements on the Contiki-NG and lwIP 2.1.2 TCP implementations. He also provided details on the uIP TCP implementation.

7. 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.

7.1. uIP

uIP is a TCP/IP stack, targetted 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, IPv4, ICMP and TCP) has been reported for uIP [Dunk]. Later versions of uIP implement IPv6 as well.

uIP uses the same global buffer for both incoming and outgoing traffic, which 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. Multiple connections are supported, but need to share the global buffer.

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 3.2.3) to avoid Delayed ACKs for senders using a single segment.

The code size of the TCP implementation in Contiki-NG has been measured to be of 3.2 kB on CC2538DK, cross-compiling on Linux.

7.2. lwIP

lwIP is a TCP/IP stack, targetted 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, IPv4, ICMP and TCP), and a TCP code size of ~9 kB to ~14 kB [Dunk]. Both IPv4 and IPv6 are supported in lwIP since v2.0.0.

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 outcoming data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit and fast recovery. SACK and Window Scale support has 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, with 32-bit platforms also supported. 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. For simplicity, GNRC TCP uses a single-MSS 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 offers an optional POSIX socket wrapper that enables POSIX compliance, if needed.

Further details on RIOT and GNRC can be found in the literature [RIOT], [GNRC].

7.4. TinyOS

TinyOS was important as a 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 by the application. Multiple TCP connections are possible. Recently there has been little further work on the stack.

7.5. 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-segment 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'.

7.6. 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-segment window size.

7.7. Summary

		uIP	lwIP orig	lwIP 2.1	RIOT	TinyOS	FreeRTOS	uC/OS
Memory	Code size(kB)	<5 (a)	~9 to ~14 (T1)	38 (T4)	<7 (T3)	N/A	<9.2 (T2)	N/A
TCP features	Single-Segm.	Yes	No	No	Yes	No	No	No
	Slow start	No	Yes	Yes	No	Yes	No	Yes
	Fast rec/retx	No	Yes	Yes	No	Yes	No	Yes
	Keep-alive	No	No	Yes	No	No	Yes	Yes
	Win. Scale	No	No	Yes	No	No	Yes	No
	TCP timest.	No	No	Yes	No	No	Yes	No
	SACK	No	No	Yes	No	No	Yes	No
	Del. ACKs	No	Yes	Yes	No	No	Yes	Yes
	Socket	No	No	Optional	(I)	Subset	Yes	Yes
	Concur. Conn.	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TLS supported		No	No	Yes	Yes	Yes	Yes	Yes

(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)
(T4) = TCP-only, on CC2538DK, cross-compiling on Linux
(a) = includes IP, ICMP and TCP on x86 and AVR platforms. The Contiki-NG TCP implementation has a code size of 3.2 kB on CC2538DK, cross-compiling on Linux
(I) = optional POSIX socket wrapper which enables POSIX compliance if needed
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.

8. Annex. Changes compared to previous versions

RFC Editor: To be removed prior to publication

8.1. 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 patters
- 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

8.2. 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

8.3. 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

8.4. Changes between -03 and -04

- Addressing the remaining TODOs
- Alignment of the wording on TCP "keep-alives" with related discussions in the IETF transport area
- Added further discussion on delayed ACKs
- Removed OpenWSN section from the Annex

8.5. Changes between -04 and -05

- Addressing comments by Yoshifumi Nishida
- Removed mentioning MD5 as an example (comment by David Black)
- Added memory footprint details of TCP implementations (Contiki-NG and lwIP 2.1.2) provided by Rahul Jadhav in the Annex
- Addressed comments by Ilpo Jarvinen throughout the whole document
- Improved the RIOT section in the Annex, based on feedback from Emmanuel Baccelli

8.6. Changes between -05 and -06

- Incorporated suggestions by Stuart Cheshire

8.7. Changes between -06 and -07

- Addressed comments by Gorry Fairhurst

8.8. Changes between -07 and -08

- Addressed WGLC comments by Ilpo Jarvinen, Markku Kojo and Ingemar Johansson throughout the document, including the addition of a new section on Initial Window considerations.

8.9. Changes between -08 and -09

- Addressed second round of comments by Ilpo Jarvinen and Markku Kojo, based on the previous draft update.

8.10. Changes between -09 and -10

- Addressed comments by Erik Kline.
- Addressed a comment by Markku Kojo on advice given in RFC 6691.

8.11. Changes between -10 and -11

- Addressed a comment by Ted Lemon on MSS advice.

8.12. Changes between -11 and -12

- Addressed comments from IESG and various directorates.

8.13. Changes between -12 and -13

- Fixed two typos.
- Addressed a comment by Barry Leiba.

9. References

9.1. Normative References

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

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

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

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

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

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

[RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, DOI 10.17487/RFC6691, July 2012, <https://www.rfc-editor.org/info/rfc6691>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

9.2. Informative References

[Commag] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP Congestion Control for the Internet of Things", IEEE Communications Magazine, June 2016.

[Dunk] A. Dunkels, "Full TCP/IP for 8-Bit Architectures", 2003.

[ETEN] R. Krishnan et al, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks", Computer Networks 2004.

[GNRC] M. Lenders et al., "Connecting the World of Embedded Mobiles: The RIOTApproach to Ubiquitous Networking for the IoT", 2018.

[HomeGateway] Haetoenen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and M. Kojo, "An Experimental Study of Home Gateway Characteristics", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement 2010.

[I-D.delcarpio-6lo-wlanah] Vega, L., Robles, I., and R. Morabito, "IPv6 over 802.11ah", draft-delcarpio-6lo-wlanah-01 (work in progress), October 2015.

[I-D.ietf-6lo-fragment-recovery] Thubert, P., "6LoWPAN Selective Fragment Recovery", draft-ietf-6lo-fragment-recovery-21 (work in progress), March 2020.

[I-D.ietf-core-fasor] Jarvinen, I., Kojo, M., Raitahila, I., and Z. Cao, "Fast-Slow Retransmission Timeout and Congestion Control Algorithm for CoAP", draft-ietf-core-fasor-01 (work in progress), October 2020.

[I-D.ietf-tcpm-generalized-ecn] Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-05 (work in progress), November 2019.

[I-D.ietf-tcpm-rto-consider] Allman, M., "Requirements for Time-Based Loss Detection", draft-ietf-tcpm-rto-consider-17 (work in progress), July 2020.

[IntComp] C. Gomez, A. Arcia-Moret, J. Crowcroft, "TCP in the Internet of Things: from ostracism to prominence", IEEE Internet Computing, January-February 2018.

[MQTT] ISO/IEC 20922:2016, "Message Queuing Telemetry Transport (MQTT) v3.1.1", 2016.

[RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, DOI 10.17487/RFC2757, January 2000, <https://www.rfc-editor.org/info/rfc2757>.

[RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <https://www.rfc-editor.org/info/rfc2884>.

[RFC3481] Inamura, H., Ed., Montenegro, G., Ed., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", BCP 71, RFC 3481, DOI 10.17487/RFC3481, February 2003, <https://www.rfc-editor.org/info/rfc3481>.

[RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, DOI 10.17487/RFC3819, July 2004, <https://www.rfc-editor.org/info/rfc3819>.

[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, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6077] Papadimitriou, D., Ed., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, DOI 10.17487/RFC6077, February 2011, <https://www.rfc-editor.org/info/rfc6077>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6606] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing", RFC 6606, DOI 10.17487/RFC6606, May 2012, <https://www.rfc-editor.org/info/rfc6606>.

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <https://www.rfc-editor.org/info/rfc7414>.

[RFC7428] Brandt, A. and J. Buron, "Transmission of IPv6 Packets over ITU-T G.9959 Networks", RFC 7428, DOI 10.17487/RFC7428, February 2015, <https://www.rfc-editor.org/info/rfc7428>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <https://www.rfc-editor.org/info/rfc7668>.

[RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <https://www.rfc-editor.org/info/rfc8105>.

[RFC8163] Lynn, K., Ed., Martocci, J., Neilson, C., and S. Donaldson, "Transmission of IPv6 over Master-Slave/Token-Passing (MS/TP) Networks", RFC 8163, DOI 10.17487/RFC8163, May 2017, <https://www.rfc-editor.org/info/rfc8163>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[RFC8352] Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, Ed., "Energy-Efficient Features of Internet of Things Protocols", RFC 8352, DOI 10.17487/RFC8352, April 2018, <https://www.rfc-editor.org/info/rfc8352>.

[RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <https://www.rfc-editor.org/info/rfc8376>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <https://www.rfc-editor.org/info/rfc8900>.

[RIOT] E. Baccelli et al., "RIOT: an Open Source Operating Systemfor Low-end Embedded Devices in the IoT", 2018.

[shrew] A. Kuzmanovic, E. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks", SIGCOMM'03 2003.

Authors' Addresses

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

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

Michael Scharf
Hochschule Esslingen
Flandernstr. 101
Esslingen 73732
Germany

Email: michael.scharf@hs-esslingen.de

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 26, 2018

J. Mattsson
F. Palombini
Ericsson AB
February 22, 2018

Comparison of CoAP Security Protocols
draft-mattsson-lwig-security-protocol-comparison-00

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 anlyzed with and without Connection ID.

Table of Contents

1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UPD [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 anlyzed 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 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 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 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.

Sequence Number	'05'	'1005'	'100005'
DTLS 1.2	29	29	29
DTLS 1.3	16	16	16
DTLS 1.3 (short header)	11	11	11
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	17	17	17
DTLS 1.3 (short header) (GCH)	12	12	12
TLS 1.2	21	21	21
TLS 1.3	14	14	14
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	15	16	17
OSCORE Request	13	14	15
OSCORE Response	11	11	11

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

Connection/Sender ID	''	'42'	'4002'
DTLS 1.2	29	30	31
DTLS 1.3	16	17	18
DTLS 1.3 (short header)	11	12	13
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	17	18	19
DTLS 1.3 (short header) (GCH)	12	13	14
OSCORE Request	13	14	15
OSCORE Response	11	11	11

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

Protocol	Overhead	Overhead (GHC)
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]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-08 (work in progress), January 2018.

[I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "The Datagram Transport Layer Security (DTLS) Connection Identifier", draft-ietf-tls-dtls-connection-id-00 (work in progress), December 2017.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-22 (work in progress), November 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-23 (work in progress), January 2018.

[OlegHahm-ghc]
Hahm, O., "Generic Header Compression", July 2016, <https://github.com/OlegHahm/ghc>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <https://www.rfc-editor.org/info/rfc7400>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 20, 2018

J. Mattsson
F. Palombini
Ericsson AB

March 19, 2018

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.

Table of Contents

1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UPD [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 anlyzed 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 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 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.

Sequence Number	'05'	'1005'	'100005'
DTLS 1.2	29	29	29
DTLS 1.3	16	16	16
DTLS 1.3 (short header)	11	11	11
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	17	17	17
DTLS 1.3 (short header) (GCH)	12	12	12
TLS 1.2	21	21	21
TLS 1.3	14	14	14
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	15	16	17
OSCORE Request	13	14	15
OSCORE Response	11	11	11

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

Connection/Sender ID	''	'42'	'4002'
DTLS 1.2	29	30	31
DTLS 1.3	16	17	18
DTLS 1.3 (short header)	11	12	13
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	17	18	19
DTLS 1.3 (short header) (GCH)	12	13	14
OSCORE Request	13	14	15
OSCORE Response	11	11	11

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

Protocol	Overhead	Overhead (GHC)
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. 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]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-11 (work in progress), March 2018.

[I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "The Datagram Transport Layer Security (DTLS) Connection Identifier", draft-ietf-tls-dtls-connection-id-00 (work in progress), December 2017.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-26 (work in progress), March 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-27 (work in progress), March 2018.

[OlegHahm-ghc]
Hahm, O., "Generic Header Compression", July 2016, <https://github.com/OlegHahm/ghc>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <https://www.rfc-editor.org/info/rfc7400>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

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

lwig
Internet-Draft
Intended status: Informational
Expires: May 17, 2018

R. Struik
Struik Security Consultancy

November 13, 2017

Alternative Elliptic Curve Representations
draft-struik-lwig-curve-representations-00

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

Table of Contents

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

[ANSI-X9.62]
ANSI X9.62-2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", American National Standard for Financial Services, Accredited Standards Committee X9, Inc Anapolis, MD, 2005.

[FIPS-186-4]
FIPS 186-4, "Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-4", US Department of Commerce/National Institute of Standards and Technology Gaithersburg, MD, July 2013.

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

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[SP-800-56a] NIST SP 800-56a, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Log Cryptography, Revision 2", US Department of Commerce/National Institute of Standards and Technology Gaithersburg, MD, June 2013.

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 P1:=(x1, y1) and P2:=(x2, y2) be distinct points on the Weierstrass curve W_{a,b} and let Q:=P1 + P2, where Q is not the identity element. Then Q:=(x, y), where

> x + x1 + x2 = lambda^2 and y + y1 = lambda*(x1 - x), where lambda = (y2 - y1)/(x2 - x1).

Let P:= (x1, y1) 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 + 2*x1 = lambda^2 and y + y1 = lambda*(x1 - x), where lambda=(3*x1^2 + a)/(2*y1).

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 P1:=(x1, y1) and P2:=(x2, y2) be distinct points on the Montgomery curve M_{A,B} and let Q:=P1 + P2, where Q is not the identity element. Then Q:=(x, y), where

x + x1 + x2 = B*lambda^2 - A and y + y1 = lambda*(x1 - x), where lambda=(y2 - y1)/(x2 - x1).

Let P:= (x1, y1) 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 + 2*x1 = B*lambda^2 - A and y + y1 = lambda*(x1 - x), where lambda=(3*x1^2 + 2*A*x1+1)/(2*y1).

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 P1:=(x1, y1) and P2:=(x2, y2) be points on the twisted Edwards curve E_{a,d} and let Q:=P1 + P2. Then Q:=(x, y), where

x = (x1*y2 + x2*y1)/(1 + d*x1*x2*y1*y2) and y = (y1*y2 - a*x1*x2)/(1 - d*x1*x2*y1*y2).

Let P:=(x1, y1) be a point on the twisted Edwards curve E_{a,d} and let Q:=2P. Then Q:=(x, y), where

x = (2*x1*y1)/(1 + d*x1^2*y1^2) and y = (y1^2 - a*x1^2)/(1 - d*x1^2*y1^2).

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 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 GF(p) defined by

delta 19298681539552699237261830834781317975544997444273427339909597334652188435537

(=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2451)

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 GF(p) defined by

c sqrt(-(A+2))

510425693991605361302061352331463292841522022530346318226818337886668772152 07

(=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. (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

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffed)

h 8

n 7237005577332262213973186563042994240857116359379907606001950938285454250989

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

h1 4

n1 14474011154664524427946373126085988481603263447650325797860494125
407373907997

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

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 (-0x01)

d -121665/121666

(=37095705934669439343138083508754565189542113879843219016388785533085940283555)

(=0x52036cee 2b6ffe73 8cc74079 7779e898 00700a4d 4141d8ab
75eb4dca 135978a3)

Gx 15112221349535400772501151409588531511454012693041857206046113283
949847762202

(=0x216936d3 cd6e53fe c0a4e231 fdd6dc5c 692cc760 9525a7b2
c9562d60 8f25d51a)

Gy 4/5

(=46316835694926478169428394003475163141307993866256225615783033603162512518559 60)

(=0x66666666 66666666 66666666 66666666 66666666 66666666
66666666 66666658)

Weierstrass curve-specific parameters (for Wei25519):

a 19298681539552699237261830834781317975544997444273427339909597334
573241639236

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaa98 4914a144)

b 55751746669818908907645289078257140818241103727901012315294400837
956729358436

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

Gx’ 19298681539552699237261830834781317975544997444273427339909597334
652188435546

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaaaa 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

lwig
Internet-Draft
Intended status: Informational
Expires: January 20, 2019

R. Struik
Struik Security Consultancy

July 19, 2018

Alternative Elliptic Curve Representations
draft-struik-lwig-curve-representations-02

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

Table of Contents

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

[ANSI-X9.62]
ANSI X9.62-2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", American National Standard for Financial Services, Accredited Standards Committee X9, Inc Anapolis, MD, 2005.

[FIPS-186-4]
FIPS 186-4, "Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-4", US Department of Commerce/National Institute of Standards and Technology Gaithersburg, MD, July 2013.

[GECC] D. Hankerson, A.J. Menezes, S.A. Vanstone, "Guide to Elliptic Curve Cryptography", New York: Springer-Verlag, 2004.

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

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, DOI 10.17487/RFC5639, March 2010, <https://www.rfc-editor.org/info/rfc5639>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[SP-800-56a]
NIST SP 800-56a, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Log Cryptography, Revision 2", US Department of Commerce/National Institute of Standards and Technology Gaithersburg, MD, June 2013.

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 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*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 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*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 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 P1:=(x1, y1) and P2:=(x2, y2) be distinct affine points of the Weierstrass curve W_{a,b} and let Q:=P1 + P2, where Q is not the identity element. Then Q:=(x, y), where

> x + x1 + x2 = lambda^2 and y + y1 = lambda*(x1 - x), where lambda = (y2 - y1)/(x2 - x1).

Let P:= (x1, y1) 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 + 2*x1 = lambda^2 and y + y1 = lambda*(x1 - x), where lambda=(3*x1^2 + a)/(2*y1).

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 P1:=(x1, y1) and P2:=(x2, y2) be distinct affine points of the Montgomery curve M_{A,B} and let Q:=P1 + P2, where Q is not the identity element. Then Q:=(x, y), where

> x + x1 + x2 = B*lambda^2 - A and y + y1 = lambda*(x1 - x), where lambda=(y2 - y1)/(x2 - x1).

Let P:= (x1, y1) 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 + 2*x1 = B*lambda^2 - A and y + y1 = lambda*(x1 - x), where lambda=(3*x1^2 + 2*A*x1+1)/(2*y1).

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 P1:=(x1, y1) and P2:=(x2, y2) be points of the twisted Edwards curve E_{a,d} and let Q:=P1 + P2. Then Q:=(x, y), where

$$x = (x1*y2 + x2*y1)/(1 + d*x1*x2*y1*y2) \text{ and } y = (y1*y2 - a*x1*x2)/(1 - d*x1*x2*y1*y2).$$

Let P:=(x1, y1) be a point of the twisted Edwards curve E_{a,d} and let Q:=2P. Then Q:=(x, y), where

$$x = (2*x1*y1)/(1 + d*x1^2*y1^2) \text{ and } y = (y1^2 - a*x1^2)/(1 - d*x1^2*y1^2).$$

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 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. 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 GF(p) defined by

delta 19298681539552699237261830834781317975544997444273427339909597334652188435537

(=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2451)

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 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. (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

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffed)

h 8

n 7237005577332262213973186563042994240857116359379907606001950938285454250989

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

h1 4

n1 1447401115466452442794637312608598848160326344765032579786049412540737390799 7

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

Montgomery curve-specific parameters (for Curve25519):

A 486662

B 1

Gu 9 (=0x9)

Gv 1478161944758954479102059356840998688726460613461647528896488183775558623740 1

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

Twisted Edwards curve-specific parameters (for Edwards25519):

a -1 (-0x01)

d -121665/121666

(=37095705934669439343138083508754565189542113879843219016388785533085940283555)

(=0x52036cee 2b6ffe73 8cc74079 7779e898 00700a4d 4141d8ab
75eb4dca 135978a3)

Gx 15112221349535400772501151409588531511454012693041857206046113283949847762202

(=0x216936d3 cd6e53fe c0a4e231 fdd6dc5c 692cc760 9525a7b2
c9562d60 8f25d51a)

Gy 4/5

(=46316835694926478169428394003475163141307993866256225615783033603162518559600)

(=0x66666666 66666666 66666666 66666666 66666666 66666666
66666666 66666658)

Weierstrass curve-specific parameters (for Wei25519):

a 19298681539552699237261830834781317975544997444273427339909597334573241639236

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaa98 4914a144)

b 55751746669818908907645289078257140818241103727901012315294400837956729358436

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

Gx’ 19298681539552699237261830834781317975544997444273427339909597334652188435546

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaaaa aaad245a)

Gy' 14781619447589544791020593568409986887264606134616475288964881837
755586237401

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

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 (mod 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 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 GF(q). However, even in this case, one can still express the curve 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 W_{a,b} to points of the Weierstrass curve W_{a',b'}, where a':=-3 (mod p), even if a'/a is not a fourth power in GF(q). In that case, this mappping 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 4773168724887355967255521690649675419508341069991820702939107936363
6321486119

(=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):

a 2 (=0x2)

b 45793404337388339159414415854563976158160282736335993851976016290777777599260

(=0x653e25fa 4aa43eb9 cc42c61b 806bcfd1 0e67bc23 09966e90 95a202fe 9aac731c)

G1x' 21872607226894442744132797191435288341483620396057247222462149535754145422686

(=0x305b74fc 935f1dad d440a88e 781f0a81 09d6a68d 98c6081a 660528e2 0746dd5e)

G1y' 13943617903486429134407723576638679615598775530747991987186632147013341290929

(=0x1ed3cedc e78b6b19 5d1c361c e1d4ef00 5b5b102c 99083780 bf830f7e a89021b1)

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

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

a -3

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff fffffffea)

b [TBD]

(=0x[TBD])

G2x’ [TBD]

(=0x[TBD])

G2y’ [TBD]

(=0x[TBD])

Author’s Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com