

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

C. Bormann
Universitaet Bremen TZI
M. Ersue
Nokia Solutions and Networks
A. Keranen
Ericsson
C. Gomez
UPC/i2CAT
July 02, 2018

Terminology for Constrained-Node Networks
draft-bormann-lwig-7228bis-03

Abstract

The Internet Protocol Suite is increasingly used on small devices with severe constraints on power, memory, and processing resources, creating constrained-node networks. This document provides a number of basic terms that have been useful in the standardization work for constrained-node networks.

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 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
2. Core Terminology	3
2.1. Constrained Nodes	4
2.2. Constrained Networks	5
2.2.1. Challenged Networks	6
2.3. Constrained-Node Networks	6
2.3.1. LLN	7
2.3.2. LoWPAN, 6LoWPAN	7
3. Classes of Constrained Devices	8
4. Power Terminology	11
4.1. Scaling Properties	11
4.2. Classes of Energy Limitation	12
4.3. Strategies for Using Power for Communication	13
5. Classes of Networks	15
5.1. Classes of link layer MTU size	15
5.2. Class of Internet Integration	16
5.3. Classes of physical layer bit rate	16
6. IANA Considerations	18
7. Security Considerations	18
8. Informative References	18
Acknowledgements	22
Authors' Addresses	22

1. Introduction

Small devices with limited CPU, memory, and power resources, so-called "constrained devices" (often used as sensors/actuators, smart objects, or smart devices) can form a network, becoming "constrained nodes" in that network. Such a network may itself exhibit constraints, e.g., with unreliable or lossy channels, limited and unpredictable bandwidth, and a highly dynamic topology.

Constrained devices might be in charge of gathering information in diverse settings, including natural ecosystems, buildings, and factories, and sending the information to one or more server stations. They might also act on information, by performing some physical action, including displaying it. Constrained devices may work under severe resource constraints such as limited battery and computing power, little memory, and insufficient wireless bandwidth and ability to communicate; these constraints often exacerbate each

other. Other entities on the network, e.g., a base station or controlling server, might have more computational and communication resources and could support the interaction between the constrained devices and applications in more traditional networks.

Today, diverse sizes of constrained devices with different resources and capabilities are becoming connected. Mobile personal gadgets, building-automation devices, cellular phones, machine-to-machine (M2M) devices, and other devices benefit from interacting with other "things" nearby or somewhere in the Internet. With this, the Internet of Things (IoT) becomes a reality, built up out of uniquely identifiable and addressable objects (things). Over the next decade, this could grow to large numbers [FIFTY-BILLION] of Internet-connected constrained devices, greatly increasing the Internet's size and scope.

The present document provides a number of basic terms that have been useful in the standardization work for constrained environments. The intention is not to exhaustively cover the field but to make sure a few core terms are used consistently between different groups cooperating in this space.

The present document is an update of [RFC7228].

In this document, the term "byte" is used in its now customary sense as a synonym for "octet". Where sizes of semiconductor memory are given, the prefix "kibi" (1024) is combined with "byte" to "kibibyte", abbreviated "KiB", for 1024 bytes [ISQ-13].

In computing, the term "power" is often used for the concept of "computing power" or "processing power", as in CPU performance. In this document, the term stands for electrical power unless explicitly stated otherwise. "Mains-powered" is used as a shorthand for being permanently connected to a stable electrical power grid.

2. Core Terminology

There are two important aspects to _scaling_ within the Internet of Things:

- o scaling up Internet technologies to a large number [FIFTY-BILLION] of inexpensive nodes, while
- o scaling down the characteristics of each of these nodes and of the networks being built out of them, to make this scaling up economically and physically viable.

The need for scaling down the characteristics of nodes leads to "constrained nodes".

2.1. Constrained Nodes

The term "constrained node" is best defined by contrasting the characteristics of a constrained node with certain widely held expectations on more familiar Internet nodes:

Constrained Node: A node where some of the characteristics that are otherwise pretty much taken for granted for Internet nodes at the time of writing are not attainable, often due to cost constraints and/or physical constraints on characteristics such as size, weight, and available power and energy. The tight limits on power, memory, and processing resources lead to hard upper bounds on state, code space, and processing cycles, making optimization of energy and network bandwidth usage a dominating consideration in all design requirements. Also, some layer-2 services such as full connectivity and broadcast/multicast may be lacking.

While this is not a rigorous definition, it is grounded in the state of the art and clearly sets apart constrained nodes from server systems, desktop or laptop computers, powerful mobile devices such as smartphones, etc. There may be many design considerations that lead to these constraints, including cost, size, weight, and other scaling factors.

(An alternative term, when the properties as a network node are not in focus, is "constrained device".)

There are multiple facets to the constraints on nodes, often applying in combination, for example:

- o constraints on the maximum code complexity (ROM/Flash),
- o constraints on the size of state and buffers (RAM),
- o constraints on the amount of computation feasible in a period of time ("processing power"),
- o constraints on the available power, and
- o constraints on user interface and accessibility in deployment (ability to set keys, update software, etc.).

Section 3 defines a small number of interesting classes ("class-N" for N = 0, 1, 2) of constrained nodes focusing on relevant combinations of the first two constraints. With respect to available

power, [RFC6606] distinguishes "power-affluent" nodes (mains-powered or regularly recharged) from "power-constrained nodes" that draw their power from primary batteries or by using energy harvesting; more detailed power terminology is given in Section 4.

The use of constrained nodes in networks often also leads to constraints on the networks themselves. However, there may also be constraints on networks that are largely independent from those of the nodes. We therefore distinguish "constrained networks" from "constrained-node networks".

2.2. Constrained Networks

We define "constrained network" in a similar way:

Constrained Network: A network where some of the characteristics pretty much taken for granted with link layers in common use in the Internet at the time of writing are not attainable.

Constraints may include:

- o low achievable bitrate/throughput (including limits on duty cycle),
- o high packet loss and high variability of packet loss (delivery rate),
- o highly asymmetric link characteristics,
- o severe penalties for using larger packets (e.g., high packet loss due to link-layer fragmentation),
- o limits on reachability over time (a substantial number of devices may power off at any point in time but periodically "wake up" and can communicate for brief periods of time), and
- o lack of (or severe constraints on) advanced services such as IP multicast.

More generally, we speak of constrained networks whenever at least some of the nodes involved in the network exhibit these characteristics.

Again, there may be several reasons for this:

- o cost constraints on the network,
- o constraints posed by the nodes (for constrained-node networks),

- o physical constraints (e.g., power constraints, environmental constraints, media constraints such as underwater operation, limited spectrum for very high density, electromagnetic compatibility),
- o regulatory constraints, such as very limited spectrum availability (including limits on effective radiated power and duty cycle) or explosion safety, and
- o technology constraints, such as older and lower-speed technologies that are still operational and may need to stay in use for some more time.

2.2.1. Challenged Networks

A constrained network is not necessarily a "challenged network" [FALL]:

Challenged Network: A network that has serious trouble maintaining what an application would today expect of the end-to-end IP model, e.g., by:

- * not being able to offer end-to-end IP connectivity at all,
- * exhibiting serious interruptions in end-to-end IP connectivity, or
- * exhibiting delay well beyond the Maximum Segment Lifetime (MSL) defined by TCP [RFC0793].

All challenged networks are constrained networks in some sense, but not all constrained networks are challenged networks. There is no well-defined boundary between the two, though. Delay-Tolerant Networking (DTN) has been designed to cope with challenged networks [RFC4838].

2.3. Constrained-Node Networks

Constrained-Node Network: A network whose characteristics are influenced by being composed of a significant portion of constrained nodes.

A constrained-node network always is a constrained network because of the network constraints stemming from the node constraints, but it may also have other constraints that already make it a constrained network.

The rest of this subsection introduces two additional terms that are in active use in the area of constrained-node networks, without an intent to define them: LLN and (6)LoWPAN.

2.3.1. LLN

A related term that has been used to describe the focus of the IETF ROLL working group is "Low-Power and Lossy Network (LLN)". The ROLL (Routing Over Low-Power and Lossy) terminology document [RFC7102] defines LLNs as follows:

LLN: Low-Power and Lossy Network. Typically composed of many embedded devices with limited power, memory, and processing resources interconnected by a variety of links, such as IEEE 802.15.4 or low-power Wi-Fi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (heating, ventilation, and air conditioning (HVAC), lighting, access control, fire), connected home, health care, environmental monitoring, urban sensor networks, energy management, assets tracking, and refrigeration.

Beyond that, LLNs often exhibit considerable loss at the physical layer, with significant variability of the delivery rate, and some short-term unreliability, coupled with some medium-term stability that makes it worthwhile to both construct directed acyclic graphs that are medium-term stable for routing and do measurements on the edges such as Expected Transmission Count (ETX) [RFC6551]. Not all LLNs comprise low-power nodes [I-D.hui-vasseur-roll-rpl-deployment].

LLNs typically are composed of constrained nodes; this leads to the design of operation modes such as the "non-storing mode" defined by RPL (the IPv6 Routing Protocol for Low-Power and Lossy Networks [RFC6550]). So, in the terminology of the present document, an LLN is a constrained-node network with certain network characteristics, which include constraints on the network as well.

2.3.2. LoWPAN, 6LoWPAN

One interesting class of a constrained network often used as a constrained-node network is "LoWPAN" [RFC4919], a term inspired from the name of an IEEE 802.15.4 working group (low-rate wireless personal area networks (LR-WPANs)). The expansion of the LoWPAN acronym, "Low-Power Wireless Personal Area Network", contains a hard-to-justify "Personal" that is due to the history of task group naming in IEEE 802 more than due to an orientation of LoWPANs around a single person. Actually, LoWPANs have been suggested for urban monitoring, control of large buildings, and industrial control applications, so the "Personal" can only be considered a vestige.

Occasionally, the term is read as "Low-Power Wireless Area Networks" [WEI]. Originally focused on IEEE 802.15.4, "LoWPAN" (or when used for IPv6, "6LoWPAN") also refers to networks built from similarly constrained link-layer technologies [RFC7668] [RFC8105] [RFC7428].

3. Classes of Constrained Devices

Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may be worthwhile to have some succinct terminology for different classes of constrained devices.

Before we get to that, let's first distinguish two big rough groups of devices based on their CPU capabilities:

- o Microcontroller-class devices (ARM term: "M-class" [need ref]). These often (but not always) include RAM and code storage on chip and limit their support for general-purpose operating systems, e.g., they do not have an MMU (memory management unit). They use most of their pins for interfaces to application hardware such as digital in/out (the latter often PWM-controllable), ADC/DACs, etc. Where this hardware is specialized for an application, we may talk about "Systems on a Chip" (SOC). These devices often implement elaborate sleep modes to achieve microwatt- or at least milliwatt-level sustained power usage (Ps, see below).
- o General-purpose-class devices (ARM term: "A-class"). These usually have RAM and Flash storage on separate chips (not always separate packages), and offer support for general-purpose operating systems such as Linux, e.g. an MMU. Many of the pins on the CPU chip are dedicated to interfacing with RAM and other memory. Some general-purpose-class devices integrate some application hardware such as video controllers, these are often called "Systems on a Chip" (SOC). While these chips also include sleep modes, they are usually more on the watt side of sustained power usage (Ps).

If the distinction between these groups needs to be made in this document, we distinguish group "M" (microcontroller) from group "J" (general purpose).

In this document, the class designations in Table 1 may be used as rough indications of device capabilities. Note that the classes from 10 upwards are not really constrained devices in the sense of the previous section; they may still be useful to discuss constraints in larger devices:

Group	Name	data size (e.g., RAM)	code size (e.g., Flash)	Examples
M	Class 0, C0	<< 10 KiB	<< 100 KiB	ATtiny
M	Class 1, C1	~ 10 KiB	~ 100 KiB	STM32F103CB
M	Class 2, C2	~ 50 KiB	~ 250 KiB	STM32F103RC
M	Class 3, C3	~ 100 KiB	~ 500..1000 KiB	STM32F103RG
M	Class 4, C4	~ 300..500..1000 KiB	~ 1000...2000 KiB	"Luxury"
J	Class 10, C10	4-8 MiB	(?)	OpenWRT routers
J		fill in useful	J-group classes	
J	Class 13, C13	0.5..1 GiB	(lots)	Raspberry PI
J	Class 15, C15	1..2 GiB	(lots)	Smartphones
J	Class 16, C16	4..32 GiB	(lots)	Laptops
J	Class 19, C19	(lots)	(lots)	Servers

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

As of the writing of this document, these characteristics correspond to distinguishable clusters of commercially available chips and design cores for constrained devices. While it is expected that the boundaries of these classes will move over time, Moore's law tends to be less effective in the embedded space than in personal computing devices: gains made available by increases in transistor count and density are more likely to be invested in reductions of cost and power requirements than into continual increases in computing power.

Class 0 devices are very constrained sensor-like nodes. They are so severely constrained in memory and processing capabilities that most likely they will not have the resources required to communicate directly with the Internet in a secure manner (rare heroic, narrowly targeted implementation efforts notwithstanding). Class 0 devices will participate in Internet communications with the help of larger devices acting as proxies, gateways, or servers. Class 0 devices generally cannot be secured or managed comprehensively in the traditional sense. They will most likely be preconfigured (and will be reconfigured rarely, if at all) with a very small data set. For management purposes, they could answer keepalive signals and send on/off or basic health indications.

Class 1 devices are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack such as using HTTP, Transport Layer Security (TLS), and related security protocols and XML-based data representations. However, they are capable enough to use a protocol stack specifically designed for constrained nodes (such as the Constrained Application Protocol (CoAP) over UDP [RFC7252]) and participate in meaningful conversations without the help of a gateway node. In particular, they can provide support for the security functions required on a large network. Therefore, they can be integrated as fully developed peers into an IP network, but they need to be parsimonious with state memory, code space, and often power expenditure for protocol and application usage.

Class 2 devices are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers. However, even these devices can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth. Furthermore, using fewer resources for networking leaves more resources available to applications. Thus, using the protocol stacks defined for more constrained devices on Class 2 devices might reduce development costs and increase the interoperability.

Constrained devices with capabilities significantly beyond Class 2 devices exist. They are less demanding from a standards development point of view as they can largely use existing protocols unchanged. The previous version of the present document therefore did not make any attempt to define constrained classes beyond Class 2. These devices, and to a certain extent even J-group devices, can still be constrained by a limited energy supply. Class 3 and 4 devices are less clearly defined than the lower classes; they are even less constrained. In particular Class 4 devices are powerful enough to quite comfortably run, e.g., JavaScript interpreters, together with elaborate network stacks. Additional classes may need to be defined

based on protection capabilities, e.g., an MPU (memory protection unit; true MMUs are typically only found in J-group devices).

With respect to examining the capabilities of constrained nodes, particularly for Class 1 devices, it is important to understand what type of applications they are able to run and which protocol mechanisms would be most suitable. Because of memory and other limitations, each specific Class 1 device might be able to support only a few selected functions needed for its intended operation. In other words, the set of functions that can actually be supported is not static per device type: devices with similar constraints might choose to support different functions. Even though Class 2 devices have some more functionality available and may be able to provide a more complete set of functions, they still need to be assessed for the type of applications they will be running and the protocol functions they would need. To be able to derive any requirements, the use cases and the involvement of the devices in the application and the operational scenario need to be analyzed. Use cases may combine constrained devices of multiple classes as well as more traditional Internet nodes.

4. Power Terminology

Devices not only differ in their computing capabilities but also in available power and/or energy. While it is harder to find recognizable clusters in this space, it is still useful to introduce some common terminology.

4.1. Scaling Properties

The power and/or energy available to a device may vastly differ, from kilowatts to microwatts, from essentially unlimited to hundreds of microjoules.

Instead of defining classes or clusters, we simply state, using the International System of Units (SI units), an approximate value for one or both of the quantities listed in Table 2:

Name	Definition	SI Unit
Ps	Sustainable average power available for the device over the time it is functioning	W (Watt)
Et	Total electrical energy available before the energy source is exhausted	J (Joule)

Table 2: Quantities Relevant to Power and Energy

The value of Et may need to be interpreted in conjunction with an indication over which period of time the value is given; see Section 4.2.

Some devices enter a "low-power" mode before the energy available in a period is exhausted or even have multiple such steps on the way to exhaustion. For these devices, Ps would need to be given for each of the modes/steps.

4.2. Classes of Energy Limitation

As discussed above, some devices are limited in available energy as opposed to (or in addition to) being limited in available power. Where no relevant limitations exist with respect to energy, the device is classified as E9. The energy limitation may be in total energy available in the usable lifetime of the device (e.g., a device that is discarded when its non-replaceable primary battery is exhausted), classified as E2. Where the relevant limitation is for a specific period, the device is classified as E1, e.g., a solar-powered device with a limited amount of energy available for the night, a device that is manually connected to a charger and has a period of time between recharges, or a device with a periodic (primary) battery replacement interval. Finally, there may be a limited amount of energy available for a specific event, e.g., for a button press in an energy-harvesting light switch; such devices are classified as E0. Note that, in a sense, many E1 devices are also E2, as the rechargeable battery has a limited number of useful recharging cycles.

Table 3 provides a summary of the classifications described above.

Name	Type of energy limitation	Example Power Source
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery that is periodically recharged or replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No direct quantitative limitations to available energy	Mains-powered

Table 3: Classes of Energy Limitation

4.3. Strategies for Using Power for Communication

Especially when wireless transmission is used, the radio often consumes a big portion of the total energy consumed by the device. Design parameters, such as the available spectrum, the desired range, and the bitrate aimed for, influence the power consumed during transmission and reception; the duration of transmission and reception (including potential reception) influence the total energy consumption.

Different strategies for power usage and network attachment may be used, based on the type of the energy source (e.g., battery or mains-powered) and the frequency with which a device needs to communicate.

The general strategies for power usage can be described as follows:

Always-on: This strategy is most applicable if there is no reason for extreme measures for power saving. The device can stay on in the usual manner all the time. It may be useful to employ power-friendly hardware or limit the number of wireless transmissions, CPU speeds, and other aspects for general power-saving and cooling needs, but the device can be connected to the network all the time.

Normally-off: Under this strategy, the device sleeps such long periods at a time that once it wakes up, it makes sense for it to not pretend that it has been connected to the network during sleep: the device reattaches to the network as it is woken up. The main optimization goal is to minimize the effort during the

reattachment process and any resulting application communications. If the device sleeps for long periods of time and needs to communicate infrequently, the relative increase in energy expenditure during reattachment may be acceptable.

Low-power: This strategy is most applicable to devices that need to operate on a very small amount of power but still need to be able to communicate on a relatively frequent basis. This implies that extremely low-power solutions need to be used for the hardware, chosen link-layer mechanisms, and so on. Typically, given the small amount of time between transmissions, despite their sleep state, these devices retain some form of attachment to the network. Techniques used for minimizing power usage for the network communications include minimizing any work from re-establishing communications after waking up and tuning the frequency of communications (including "duty cycling", where components are switched on and off in a regular cycle) and other parameters appropriately.

Table 4 provides a summary of the strategies described above.

Name	Strategy	Ability to communicate
P0	Normally-off	Reattach when required
P1	Low-power	Appears connected, perhaps with high latency
P9	Always-on	Always connected

Table 4: Strategies of Using Power for Communication

Note that the discussion above is at the device level; similar considerations can apply at the communications-interface level. This document does not define terminology for the latter.

A term often used to describe power-saving approaches is "duty-cycling". This describes all forms of periodically switching off some function, leaving it on only for a certain percentage of time (the "duty cycle").

[RFC7102] only distinguishes two levels, defining a Non-Sleepy Node as a node that always remains in a fully powered-on state (always awake) where it has the capability to perform communication (P9) and a Sleepy Node as a node that may sometimes go into a sleep mode (a

low-power state to conserve power) and temporarily suspend protocol communication (P0); there is no explicit mention of P1.

5. Classes of Networks

5.1. Classes of link layer MTU size

Link layer technologies used by constrained devices can be categorized on the basis of link layer MTU size. Depending on this parameter, the fragmentation techniques needed (if any) to support the IPv6 MTU requirement may vary.

We define the following classes of link layer MTU size:

Name	L2 MTU size (bytes)	6LoWPAN Fragmentation applicable*?
S0	3 - 12	need new kind of fragmentation
S1	13 - 127	yes
S2	128 - 1279	yes
S2	>= 1280	no fragmentation needed

*if no link layer fragmentation is available
(note: 'Sx' stands for 'Size x')

S0 technologies require fragmentation to support the IPv6 MTU requirement. If no link layer fragmentation is available, fragmentation is needed at the adaptation layer below IPv6. However, 6LoWPAN fragmentation [RFC4944] cannot be used for these technologies, given the extremely reduced link layer MTU. In this case, lightweight fragmentation formats must be used (e.g. [I-D.ietf-lpwan-ipv6-static-context-hc]).

S1 and S2 technologies require fragmentation at the subnetwork level to support the IPv6 MTU requirement. If link layer fragmentation is unavailable or insufficient, fragmentation is needed at the adaptation layer below IPv6. 6LoWPAN fragmentation [RFC4944] can be used to carry 1280-byte IPv6 packets over these technologies.

S3 technologies do not require fragmentation to support the IPv6 MTU requirement.

5.2. Class of Internet Integration

The term "Internet of Things" is sometimes confusingly used for connected devices that are not actually employing Internet technology. Some devices do use Internet technology, but only use it to exchange packets with a fixed communication partner ("device-to-cloud" scenarios, [RFC7452]). More general devices are prepared to communicate with other nodes in the Internet as well.

We define the following classes of Internet technology level:

Name	Internet technology
I0	none (local interconnect only)
I1	device-to-cloud only
I9	full Internet connectivity supported

5.3. Classes of physical layer bit rate

[This section is a trial balloon. We could also talk about burst rate, sustained rate; bits/s, messages/s, ...]

Physical layer technologies used by constrained devices can be categorized on the basis of physical layer (PHY) bit rate. The PHY bit rate class of a technology has important implications with regard to compatibility with existing protocols and mechanisms on the Internet, responsiveness to frame transmissions and need for header compression techniques.

We define the following classes of PHY bit rate:

Name	PHY bit rate (bit/s)	Comment
B0	< 10	Tx time of 150-byte frame > MSL
B1	10 - 10 ³	Unresponsiveness if human expects reaction to sent frame (frame size > 62.5 byte)
B2	10 ³ - 10 ⁶	Responsiveness if human expects reaction to sent frame, but header compression still needed
B3	> 10 ⁶	Header compression yields relatively low performance benefits

(note: 'Bx' stands for 'Bit rate x')

B0 technologies lead to very high transmission times, which may be close to or even greater than the Maximum Segment Lifetime (MSL) assumed on the Internet [RFC0793]. Many Internet protocols and mechanisms will fail when transmit times are greater than the MSL. B0 technologies lead to a frame transmission time greater than the MSL for a frame size greater than 150 bytes.

B1 technologies offer transmission times which are lower than the MSL (for a frame size greater than 150 bytes). However, transmission times for B1 technologies are still significant if a human expects a reaction to the transmission of a frame. With B1 technologies, the transmission time of a frame greater than 62.5 bytes exceeds 0.5 seconds, i.e. a threshold time beyond which any response or reaction to a frame transmission will appear not to be immediate [RFC5826].

B2 technologies do not incur responsiveness problems, but still benefit from using header compression techniques (e.g. [RFC6282]) to achieve performance improvements.

Over B3 technologies, the relative performance benefits of header compression are low. For example, in a duty-cycled technology offering B3 PHY bit rates, energy consumption decrease due to header compression may be comparable with the energy consumed while in a sleep interval. On the other hand, for B3 PHY bit rates, a human user will not be able to perceive whether header compression has been used or not in a frame transmission.

6. IANA Considerations

This document makes no requests of IANA.

7. Security Considerations

This document introduces common terminology that does not raise any new security issues. Security considerations arising from the constraints discussed in this document need to be discussed in the context of specific protocols. For instance, Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications of specific constraints on the security mechanisms employed. [RFC7416] provides a security threat analysis for the RPL routing protocol. Implementation considerations for security protocols on constrained nodes are discussed in [RFC7815] and [I-D.ietf-lwig-tls-minimal]. A wider view of security in constrained-node networks is provided in [I-D.irtf-t2trg-iot-secons].

8. Informative References

- [FALL] Fall, K., "A Delay-Tolerant Network Architecture for Challenged Internets", SIGCOMM 2003, DOI 10.1145/863955.863960, 2003.
- [FIFTY-BILLION] Ericsson, "More Than 50 Billion Connected Devices", Ericsson White Paper 284 23-3149 Uen, February 2011, <<http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf>>.
- [I-D.hui-vasseur-roll-rpl-deployment] Vasseur, J., Hui, J., Dasgupta, S., and G. Yoon, "RPL deployment experience in large scale networks", draft-hui-vasseur-roll-rpl-deployment-01 (work in progress), July 2012.
- [I-D.ietf-lpwan-ipv6-static-context-hc] Minaburo, A., Toutain, L., Gomez, C., and D. Barthel, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-16 (work in progress), June 2018.
- [I-D.ietf-lwig-tls-minimal] Kumar, S., Keoh, S., and H. Tschofenig, "A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks", draft-ietf-lwig-tls-minimal-01 (work in progress), March 2014.

- [I-D.irtf-t2trg-iot-secons]
Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-secons-15 (work in progress), May 2018.
- [ISQ-13] International Electrotechnical Commission, "International Standard -- Quantities and units -- Part 13: Information science and technology", IEC 80000-13, March 2008.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [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>>.
- [RFC5826] Brandt, A., Buron, J., and G. Porcu, "Home Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5826, DOI 10.17487/RFC5826, April 2010, <<https://www.rfc-editor.org/info/rfc5826>>.
- [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>>.

- [RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <<https://www.rfc-editor.org/info/rfc6551>>.
- [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>>.
- [RFC7102] Vasseur, JP., "Terms Used in Routing for Low-Power and Lossy Networks", RFC 7102, DOI 10.17487/RFC7102, January 2014, <<https://www.rfc-editor.org/info/rfc7102>>.
- [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>>.
- [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>>.
- [RFC7416] Tsao, T., Alexander, R., Dohler, M., Daza, V., Lozano, A., and M. Richardson, Ed., "A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs)", RFC 7416, DOI 10.17487/RFC7416, January 2015, <<https://www.rfc-editor.org/info/rfc7416>>.
- [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>>.
- [RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <<https://www.rfc-editor.org/info/rfc7452>>.
- [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>>.

- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [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>>.
- [WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", Wiley-Blackwell monograph, DOI 10.1002/9780470686218, ISBN 9780470747995, 2009.

Acknowledgements

TBD

Authors' Addresses

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

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

Mehmet Ersue
Nokia Solutions and Networks
St.-Martinstrasse 76
Munich 81541
Germany

Phone: +49 172 8432301
EMail: mehmet.ersue@nsn.com

Ari Keranen
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: ari.keranen@ericsson.com

Carles Gomez
UPC/i2CAT
Escola d'Enginyeria de Telecomunicacio i Aeroespacial
de Castelldefels
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Phone: +34-93-413-7206
EMail: carlesgo@entel.upc.edu

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: 7 October 2022

C. Bormann
Universität Bremen TZI
M. Ersue

A. Keranen
Ericsson
C. Gomez
Universitat Politecnica de Catalunya
5 April 2022

Terminology for Constrained-Node Networks
draft-bormann-lwig-7228bis-08

Abstract

The Internet Protocol Suite is increasingly used on small devices with severe constraints on power, memory, and processing resources, creating constrained-node networks. This document provides a number of basic terms that have been useful in the standardization work for constrained-node networks.

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 7 October 2022.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Core Terminology	3
2.1. Constrained Nodes	4
2.2. Constrained Networks	5
2.2.1. Challenged Networks	6
2.3. Constrained-Node Networks	6
2.3.1. LLN	7
2.3.2. LoWPAN, 6LoWPAN	8
2.3.3. LPWAN	8
3. Classes of Constrained Devices	8
3.1. Firmware/Software upgradability	12
3.2. Isolation functionality	13
3.3. Shielded secrets	13
4. Power Terminology	14
4.1. Scaling Properties	14
4.2. Classes of Energy Limitation	15
4.3. Strategies for Using Power for Communication	16
4.4. Strategies of Keeping Time over Power Events	17
5. Classes of Networks	20
5.1. Classes of link layer MTU size	20
5.2. Class of Internet Integration	21
5.3. Classes of physical layer bit rate	21
6. IANA Considerations	23
7. Security Considerations	23
8. Informative References	23
Acknowledgements	27
Authors' Addresses	27

1. Introduction

Small devices with limited CPU, memory, and power resources, so-called "constrained devices" (often used as sensors/actuators, smart objects, or smart devices) can form a network, becoming "constrained nodes" in that network. Such a network may itself exhibit constraints, e.g., with unreliable or lossy channels, limited and unpredictable bandwidth, and a highly dynamic topology.

Constrained devices might be in charge of gathering information in diverse settings, including natural ecosystems, buildings, and factories, and sending the information to one or more server stations. They might also act on information, by performing some

physical action, including displaying it. Constrained devices may work under severe resource constraints such as limited battery and computing power, little memory, and insufficient wireless bandwidth and ability to communicate; these constraints often exacerbate each other. Other entities on the network, e.g., a base station or controlling server, might have more computational and communication resources and could support the interaction between the constrained devices and applications in more traditional networks.

Today, diverse sizes of constrained devices with different resources and capabilities are becoming connected. Mobile personal gadgets, building-automation devices, cellular phones, machine-to-machine (M2M) devices, and other devices benefit from interacting with other "things" nearby or somewhere in the Internet. With this, the Internet of Things (IoT) becomes a reality, built up out of uniquely identifiable and addressable objects (things). Over the next decade, this could grow to large numbers of Internet-connected constrained devices ([IoT-2025] predicts that by, 2025, more than 2500 devices will be connected to the Internet per second), greatly increasing the Internet's size and scope.

The present document provides a number of basic terms that have been useful in the standardization work for constrained environments. The intention is not to exhaustively cover the field but to make sure a few core terms are used consistently between different groups cooperating in this space.

The present document is a revision of [RFC7228].

In this document, the term "byte" is used in its now customary sense as a synonym for "octet". Where sizes of semiconductor memory are given, the prefix "kibi" (1024) is combined with "byte" to "kibibyte", abbreviated "KiB", for 1024 bytes [ISQ-13]. Powers of 10 are given as 10^{100} where 100 is the exponent.

In computing, the term "power" is often used for the concept of "computing power" or "processing power", as in CPU performance. In this document, the term stands for electrical power unless explicitly stated otherwise. "Mains-powered" is used as a shorthand for being permanently connected to a stable electrical power grid.

2. Core Terminology

There are two important aspects to _scaling_ within the Internet of Things:

- * scaling up Internet technologies to a large number [IoT-2025] of inexpensive nodes, while

- * scaling down the characteristics of each of these nodes and of the networks being built out of them, to make this scaling up economically and physically viable.

The need for scaling down the characteristics of nodes leads to "constrained nodes".

2.1. Constrained Nodes

The term "constrained node" is best defined by contrasting the characteristics of a constrained node with certain widely held expectations on more familiar Internet nodes:

Constrained Node: A node where some of the characteristics that are otherwise pretty much taken for granted for Internet nodes at the time of writing are not attainable, often due to cost constraints and/or physical constraints on characteristics such as size, weight, and available power and energy. The tight limits on power, memory, and processing resources lead to hard upper bounds on state, code space, and processing cycles, making optimization of energy and network bandwidth usage a dominating consideration in all design requirements. Also, some layer-2 services such as full connectivity and broadcast/multicast may be lacking.

While this is not a rigorous definition, it is grounded in the state of the art and clearly sets apart constrained nodes from server systems, desktop or laptop computers, powerful mobile devices such as smartphones, etc. There may be many design considerations that lead to these constraints, including cost, size, weight, and other scaling factors.

(An alternative term, when the properties as a network node are not in focus, is "constrained device".)

There are multiple facets to the constraints on nodes, often applying in combination, for example:

- * constraints on the maximum code complexity (ROM/Flash),
- * constraints on the size of state and buffers (RAM),
- * constraints on the amount of computation feasible in a period of time ("processing power"),
- * constraints on the available power, and
- * constraints on user interface and accessibility in deployment (ability to set keys, update software, etc.).

Section 3 defines a number of interesting classes ("class-N") of constrained nodes focusing on relevant combinations of the first two constraints. With respect to available power, [RFC6606] distinguishes "power-affluent" nodes (mains-powered or regularly recharged) from "power-constrained nodes" that draw their power from primary batteries or by using energy harvesting; more detailed power terminology is given in Section 4.

The use of constrained nodes in networks often also leads to constraints on the networks themselves. However, there may also be constraints on networks that are largely independent of those of the nodes. We therefore distinguish "constrained networks" from "constrained-node networks".

2.2. Constrained Networks

We define "constrained network" in a similar way:

Constrained Network: A network where some of the characteristics pretty much taken for granted with link layers in common use in the Internet at the time of writing are not attainable.

Constraints may include:

- * low achievable bitrate/throughput (including limits on duty cycle),
- * high packet loss and high variability of packet loss (delivery rate),
- * highly asymmetric link characteristics,
- * severe penalties for using larger packets (e.g., high packet loss due to link-layer fragmentation),
- * limits on reachability over time (a substantial number of devices may power off at any point in time but periodically "wake up" and can communicate for brief periods of time), and
- * lack of (or severe constraints on) advanced services such as IP multicast.

More generally, we speak of constrained networks whenever at least some of the nodes involved in the network exhibit these characteristics.

Again, there may be several reasons for this:

- * cost constraints on the network,
- * constraints posed by the nodes (for constrained-node networks),
- * physical constraints (e.g., power constraints, environmental constraints, media constraints such as underwater operation, limited spectrum for very high density, electromagnetic compatibility),
- * regulatory constraints, such as very limited spectrum availability (including limits on effective radiated power and duty cycle) or explosion safety, and
- * technology constraints, such as older and lower-speed technologies that are still operational and may need to stay in use for some more time.

2.2.1. Challenged Networks

A constrained network is not necessarily a "challenged network" [FALL]:

Challenged Network: A network that has serious trouble maintaining what an application would today expect of the end-to-end IP model, e.g., by:

- * not being able to offer end-to-end IP connectivity at all,
- * exhibiting serious interruptions in end-to-end IP connectivity, or
- * exhibiting delay well beyond the Maximum Segment Lifetime (MSL) defined by TCP [RFC0793].

All challenged networks are constrained networks in some sense, but not all constrained networks are challenged networks. There is no well-defined boundary between the two, though. Delay-Tolerant Networking (DTN) has been designed to cope with challenged networks [RFC4838].

2.3. Constrained-Node Networks

Constrained-Node Network: A network whose characteristics are influenced by being composed of a significant portion of constrained nodes.

A constrained-node network always is a constrained network because of the network constraints stemming from the node constraints, but it may also have other constraints that already make it a constrained network.

The rest of this subsection introduces two additional terms that are in active use in the area of constrained-node networks, without an intent to define them: LLN and (6)LoWPAN.

2.3.1. LLN

A related term that has been used to describe the focus of the IETF ROLL working group is "Low-Power and Lossy Network (LLN)". The ROLL (Routing Over Low-Power and Lossy) terminology document [RFC7102] defines LLNs as follows:

LLN: Low-Power and Lossy Network. Typically composed of many embedded devices with limited power, memory, and processing resources interconnected by a variety of links, such as IEEE 802.15.4 or low-power Wi-Fi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (heating, ventilation, and air conditioning (HVAC), lighting, access control, fire), connected home, health care, environmental monitoring, urban sensor networks, energy management, assets tracking, and refrigeration.

Beyond that, LLNs often exhibit considerable loss at the physical layer, with significant variability of the delivery rate, and some short-term unreliability, coupled with some medium-term stability that makes it worthwhile to both (1) construct directed acyclic graphs that are medium-term stable for routing and (2) do measurements on the edges such as Expected Transmission Count (ETX) [RFC6551]. Not all LLNs comprise low-power nodes [I-D.hui-vasseur-roll-rpl-deployment].

LLNs typically are composed of constrained nodes; this leads to the design of operation modes such as the "non-storing mode" defined by RPL (the IPv6 Routing Protocol for Low-Power and Lossy Networks [RFC6550]). So, in the terminology of the present document, an LLN is a constrained-node network with certain network characteristics, which include constraints on the network as well.

2.3.2. LoWPAN, 6LoWPAN

One interesting class of a constrained network often used as a constrained-node network is "LoWPAN" [RFC4919], a term inspired from the name of an IEEE 802.15.4 working group (low-rate wireless personal area networks (LR-WPANs)). The expansion of the LoWPAN acronym, "Low-Power Wireless Personal Area Network", contains a hard-to-justify "Personal" that is due to the history of task group naming in IEEE 802 more than due to an orientation of LoWPANs around a single person. Actually, LoWPANs have been suggested for urban monitoring, control of large buildings, and industrial control applications, so the "Personal" can only be considered a vestige. Occasionally, the term is read as "Low-Power Wireless Area Networks" [WEI]. Originally focused on IEEE 802.15.4, "LoWPAN" (or when used for IPv6, "6LoWPAN") also refers to networks built from similarly constrained link-layer technologies [RFC7668] [RFC8105] [RFC7428] [RFC9159].

2.3.3. LPWAN

An overview over Low-Power Wide Area Network (LPWAN) technologies is provided by [RFC8376].

3. Classes of Constrained Devices

Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may be worthwhile to have some succinct terminology for different classes of constrained devices.

Before we get to that, let's first distinguish two big rough groups of devices based on their CPU capabilities:

- * Microcontroller-class devices (sometimes called "M-class"). These often (but not always) include RAM and code storage on chip and would struggle to support more powerful general-purpose operating systems, e.g., they do not have an MMU (memory management unit). They use most of their pins for interfaces to application hardware such as digital in/out (the latter often Pulse Width Modulation (PWM)-controllable), ADC/DACs (analog-to-digital and digital-to-analog converters), etc. Where this hardware is specialized for an application, we may talk about "Systems on a Chip" (SOC). These devices often implement elaborate sleep modes to achieve microwatt- or at least milliwatt-level sustained power usage (Ps, see below).
- * General-purpose-class devices (sometimes called "A-class"). These usually have RAM and Flash storage on separate chips (not always separate packages), and offer support for general-purpose

operating systems such as Linux, e.g. an MMU. Many of the pins on the CPU chip are dedicated to interfacing with RAM and other memory. Some general-purpose-class devices integrate some application hardware such as video controllers, these are often also called "Systems on a Chip" (SOC). While these chips also include sleep modes, they are usually more on the watt side of sustained power usage (Ps).

If the distinction between these groups needs to be made in this document, we distinguish group "M" (microcontroller) from group "J" (general purpose).

In this document, the class designations in Table 1 may be used as rough indications of device capabilities. Note that the classes from 10 upwards are not really constrained devices in the sense of the previous section; they may still be useful to discuss constraints in larger devices:

Group	Name	data size (e.g., RAM)	code size (e.g., Flash)	Examples
M	Class 0, C0	<< 10 KiB	<< 100 KiB	ATtiny
M	Class 1, C1	~ 10 KiB	~ 100 KiB	STM32F103CB
M	Class 2, C2	~ 50 KiB	~ 250 KiB	STM32F103RC
M	Class 3, C3	~ 100 KiB	~ 500..1000 KiB	STM32F103RG
M	Class 4, C4	~ 300..1000 KiB	~ 1000..2000 KiB	"Luxury"
J	Class 10, C10	(16..)32..64..128 MiB	4..8..16 MiB	OpenWRT routers
J	Class 15, C15	0.5..1 GiB	(lots)	Raspberry PI
J	Class 16, C16	1..4 GiB	(lots)	Smartphones
J	Class 17, C17	4..32 GiB	(lots)	Laptops
J	Class 19, C19	(lots)	(lots)	Servers

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

As of the writing of this document, these characteristics correspond to distinguishable clusters of commercially available chips and design cores for constrained devices. While it is expected that the boundaries of these classes will move over time, Moore's law tends to be less effective in the embedded space than in personal computing devices: gains made available by increases in transistor count and

density are more likely to be invested in reductions of cost and power requirements than into continual increases in computing power. (This effect is less pronounced in the multi-chip J-group architectures; e.g., class 10 usage for OpenWRT has started at 4/16 MiB Flash/RAM, with an early lasting minimum at 4/32, to now requiring 8/64 and preferring 16/128 for modern software releases [W432].)

Class 0 devices are very constrained sensor-like motes. They are so severely constrained in memory and processing capabilities that most likely they will not have the resources required to communicate directly with the Internet in a secure manner (rare heroic, narrowly targeted implementation efforts notwithstanding). Class 0 devices will participate in Internet communications with the help of larger devices acting as proxies, gateways, or servers. Class 0 devices generally cannot be secured or managed comprehensively in the traditional sense. They will most likely be preconfigured (and will be reconfigured rarely, if at all) with a very small data set. For management purposes, they could answer keepalive signals and send on/off or basic health indications.

Class 1 devices are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack such as using HTTP, Transport Layer Security (TLS), and related security protocols and XML-based data representations. However, they are capable enough to use a protocol stack specifically designed for constrained nodes (such as the Constrained Application Protocol (CoAP) over UDP [RFC7252]) and participate in meaningful conversations without the help of a gateway node. In particular, they can provide support for the security functions required on a large network. Therefore, they can be integrated as fully developed peers into an IP network, but they need to be parsimonious with state memory, code space, and often power expenditure for protocol and application usage.

Class 2 devices are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers. However, even these devices can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth. Furthermore, using fewer resources for networking leaves more resources available to applications. Thus, using the protocol stacks defined for more constrained devices on Class 2 devices might reduce development costs and increase the interoperability.

Constrained devices with capabilities significantly beyond Class 2 devices exist. They are less demanding from a standards development point of view as they can largely use existing protocols unchanged. The previous version of the present document therefore did not make

any attempt to define constrained classes beyond Class 2. These devices, and to a certain extent even J-group devices, can still be constrained by a limited energy supply. Class 3 and 4 devices are less clearly defined than the lower classes; they are even less constrained. In particular Class 4 devices are powerful enough to quite comfortably run, say, JavaScript interpreters, together with elaborate network stacks. Additional classes may need to be defined based on protection capabilities, e.g., an MPU (memory protection unit; true MMUs are typically only found in J-group devices).

With respect to examining the capabilities of constrained nodes, particularly for Class 1 devices, it is important to understand what type of applications they are able to run and which protocol mechanisms would be most suitable. Because of memory and other limitations, each specific Class 1 device might be able to support only a few selected functions needed for its intended operation. In other words, the set of functions that can actually be supported is not static per device type: devices with similar constraints might choose to support different functions. Even though Class 2 devices have some more functionality available and may be able to provide a more complete set of functions, they still need to be assessed for the type of applications they will be running and the protocol functions they would need. To be able to derive any requirements, the use cases and the involvement of the devices in the application and the operational scenario need to be analyzed. Use cases may combine constrained devices of multiple classes as well as more traditional Internet nodes.

3.1. Firmware/Software upgradability

Platforms may differ in their firmware or software upgradability. The below is a first attempt at classifying this.

Name	Firmware/Software upgradability
F0	no (discard for upgrade)
F1	replaceable, out of service during replacement, reboot
F2	patchable during operation, reboot required
F3	patchable during operation, restart not visible externally
F9	app-level upgradability, no reboot required ("hitless")

Table 2: Levels of software update capabilities

3.2. Isolation functionality

TBD. This section could discuss the ability of the platform to isolate different components. The categories below are not mutually exclusive; we need to build relevant clusters.

Name	Isolation functionality
Is0	no isolation
Is2	MPU (memory protection unit), at least boundary registers
Is5	MMU with Linux-style kernel/user
Is7	Virtualization-style isolation
Is8	Secure enclave isolation

Table 3: Levels of isolation capabilities

3.3. Shielded secrets

[Need to identify clusters]

Some platforms can keep shielded secrets (usually in conjunction with secure enclave functionality).

Name	Secret shielding functionality
Sh0	no secret shielding
Sh1	some secret shielding
Sh9	perfect secret shielding

Table 4: Levels of secret shielding capabilities

4. Power Terminology

Devices not only differ in their computing capabilities but also in available power and/or energy. While it is harder to find recognizable clusters in this space, it is still useful to introduce some common terminology.

4.1. Scaling Properties

The power and/or energy available to a device may vastly differ, from kilowatts to microwatts, from essentially unlimited to hundreds of microjoules.

Instead of defining classes or clusters, we simply state, using the International System of Units (SI units), an approximate value for one or both of the quantities listed in Table 5:

Name	Definition	SI Unit
Ps	Sustainable average power available for the device over the time it is functioning	W (Watt)
Et	Total electrical energy available before the energy source is exhausted	J (Joule)

Table 5: Quantities Relevant to Power and Energy

The value of Et may need to be interpreted in conjunction with an indication over which period of time the value is given; see Section 4.2.

Some devices enter a "low-power" mode before the energy available in a period is exhausted or even have multiple such steps on the way to exhaustion. For these devices, Ps would need to be given for each of the modes/steps.

4.2. Classes of Energy Limitation

As discussed above, some devices are limited in available energy as opposed to (or in addition to) being limited in available power. Where no relevant limitations exist with respect to energy, the device is classified as E9. The energy limitation may be in total energy available in the usable lifetime of the device (e.g., a device that is discarded when its non-replaceable primary battery is exhausted), classified as E2. Where the relevant limitation is for a specific period, the device is classified as E1, e.g., a solar-powered device with a limited amount of energy available for the night, a device that is manually connected to a charger and has a period of time between recharges, or a device with a periodic (primary) battery replacement interval. Finally, there may be a limited amount of energy available for a specific event, e.g., for a button press in an energy-harvesting light switch; such devices are classified as E0. Note that, in a sense, many E1 devices are also E2, as the rechargeable battery has a limited number of useful recharging cycles.

Table 6 provides a summary of the classifications described above.

Name	Type of energy limitation	Example Power Source
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery that is periodically recharged or replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No direct quantitative limitations to available energy	Mains-powered

Table 6: Classes of Energy Limitation

4.3. Strategies for Using Power for Communication

Especially when wireless transmission is used, the radio often consumes a big portion of the total energy consumed by the device. Design parameters, such as the available spectrum, the desired range, and the bitrate aimed for, influence the power consumed during transmission and reception; the duration of transmission and reception (including potential reception) influence the total energy consumption.

Different strategies for power usage and network attachment may be used, based on the type of the energy source (e.g., battery or mains-powered) and the frequency with which a device needs to communicate.

The general strategies for power usage can be described as follows:

Always-on: This strategy is most applicable if there is no reason for extreme measures for power saving. The device can stay on in the usual manner all the time. It may be useful to employ power-friendly hardware or limit the number of wireless transmissions, CPU speeds, and other aspects for general power-saving and cooling needs, but the device can be connected to the network all the time.

Normally-off: Under this strategy, the device sleeps such long periods at a time that once it wakes up, it makes sense for it to not pretend that it has been connected to the network during sleep: the device reattaches to the network as it is woken up. The main optimization goal is to minimize the effort during the reattachment process and any resulting application communications.

If the device sleeps for long periods of time and needs to communicate infrequently, the relative increase in energy expenditure during reattachment may be acceptable.

Low-power: This strategy is most applicable to devices that need to operate on a very small amount of power but still need to be able to communicate on a relatively frequent basis. This implies that extremely low-power solutions need to be used for the hardware, chosen link-layer mechanisms, and so on. Typically, given the small amount of time between transmissions, despite their sleep state, these devices retain some form of attachment to the network. Techniques used for minimizing power usage for the network communications include minimizing any work from re-establishing communications after waking up and tuning the frequency of communications (including "duty cycling", where components are switched on and off in a regular cycle) and other parameters appropriately.

Table 7 provides a summary of the strategies described above.

Name	Strategy	Ability to communicate
P0	Normally-off	Reattach when required
P1	Low-power	Appears connected, perhaps with high latency
P9	Always-on	Always connected

Table 7: Strategies of Using Power for Communication

Note that the discussion above is at the device level; similar considerations can apply at the communications-interface level. This document does not define terminology for the latter.

A term often used to describe power-saving approaches is "duty-cycling". This describes all forms of periodically switching off some function, leaving it on only for a certain percentage of time (the "duty cycle").

[RFC7102] only distinguishes two levels, defining a Non-Sleepy Node as a node that always remains in a fully powered-on state (always awake) where it has the capability to perform communication (P9) and a Sleepy Node as a node that may sometimes go into a sleep mode (a low-power state to conserve power) and temporarily suspend protocol communication (P0); there is no explicit mention of P1.

4.4. Strategies of Keeping Time over Power Events

Many applications for a device require it to keep some concept of time.

Time-keeping can be relative to a previous event (last packet received), absolute on a device-specific scale (e.g., last reboot), or absolute on a world-wide scale ("wall-clock time").

Some devices lose the concept of time when going to sleep: after wakeup, they don't know how long they slept. Some others do keep some concept of time during sleep, but not precise enough to use as a basis for keeping absolute time. Some devices have a continuously running source of a reasonably accurate time (often a 32,768 Hz watch crystal). Finally, some devices can keep their concept of time even during a battery change, e.g., by using a backup battery or a supercapacitor to power the real-time clock (RTC).

The actual accuracy of time may vary, with errors ranging from tens of percent from on-chip RC oscillators (not useful for keeping absolute time, but still useful for, e.g., timing out some state) to approximately 10^{-4} to 10^{-5} ("watch crystal") of error. More precise timing is available with temperature compensated crystal oscillators (TCXO). Further improvement requires significantly higher power usage, bulk, fragility, and device cost, e.g. oven-controlled crystal oscillators (OCXO) can reach 10^{-8} accuracy, and Rubidium frequency sources can reach 10^{-11} over the short term and 10^{-9} over the long term.

A device may need to fire up a more accurate frequency source during wireless communication, this may also allow it to keep more precise time during the period.

The various time sources available on the device can be assisted by external time input, e.g. via the network using the NTP protocol [RFC5905]. Information from measuring the deviation between external input and local time source can be used to increase the accuracy of maintaining time even during periods of no network use.

Errors of the frequency source can be compensated if known (calibrated against a known better source, or even predicted, e.g., in a software TCXO). Even with errors partially compensated, an uncertainty remains, which is the more fundamental characteristic to discuss.

Battery solutions may allow the device to keep a wall-clock time during its entire life, or the wall-clock time may need to be reset after a battery change. Even devices that have a battery lasting for their lifetime may not be set to wall-clock time at manufacture time, possibly because the battery is only activated at installation time where time sources may be questionable or because setting the clock during manufacture is deemed too much effort.

Devices that keep a good approximation of wall-clock time during their life may be in a better position to securely validate external time inputs than devices that need to be reset episodically, which can possibly be tricked by their environment into accepting a long-past time, for instance with the intent of exploiting expired security assertions such as certificates.

From a practical point of view, devices can be divided at least on the two dimensions proposed in Table 8 and Table 9. Corrections to the local time of a device performed over the network can be used to improve the uncertainty exhibited by these basic device classes.

Name	Type	Uncertainty (roughly)
T0	no concept of time	infinite
T1	relative time while awake	(usually high)
T2	relative time	(usually high during sleep)
T3	relative time	10^{-4} or better
T5	absolute time (e.g., since boot)	10^{-4} or better
T7	wall-clock time	10^{-4} or better
T8	wall-clock time	10^{-5} or better
T9	wall-clock time	10^{-6} or better (TCXO)
T10	wall-clock time	10^{-7} or better (OCXO or Rb)

Table 8: Strategies of Keeping Time over Power Events

Name	Permanency (from type T5 upwards):	Uncertainty
TP0	time needs to be reset on certain occasions	
TP1	time needs to be set during installation	(possibly reduced...
TP9	reliable time is maintained during lifetime	...by using external input)

Table 9: Permanency of Keeping Time

Further parameters that can be used to discuss clock quality can be found in Section 3.5 of [I-D.ietf-cbor-time-tag].

5. Classes of Networks

5.1. Classes of link layer MTU size

Link layer technologies used by constrained devices can be categorized on the basis of link layer MTU size. Depending on this parameter, the fragmentation techniques needed (if any) to support the IPv6 MTU requirement may vary.

We define the following classes of link layer MTU size:

Name	L2 MTU size (bytes)	6LoWPAN Fragmentation applicable*?
S0	3 - 12	need new kind of fragmentation
S1	13 - 127	yes
S2	128 - 1279	yes
S3	>= 1280	no fragmentation needed

Table 10

* if no link layer fragmentation is available (note: 'Sx' stands for 'Size x')

S0 technologies require fragmentation to support the IPv6 MTU requirement. If no link layer fragmentation is available, fragmentation is needed at the adaptation layer below IPv6. However, 6LoWPAN fragmentation [RFC4944] cannot be used for these technologies, given the extremely reduced link layer MTU. In this case, lightweight fragmentation formats must be used (e.g. [RFC8724]).

S1 and S2 technologies require fragmentation at the subnetwork level to support the IPv6 MTU requirement. If link layer fragmentation is unavailable or insufficient, fragmentation is needed at the adaptation layer below IPv6. 6LoWPAN fragmentation [RFC4944] can be used to carry 1280-byte IPv6 packets over these technologies.

S3 technologies do not require fragmentation to support the IPv6 MTU requirement.

5.2. Class of Internet Integration

The term "Internet of Things" is sometimes confusingly used for connected devices that are not actually employing Internet technology. Some devices do use Internet technology, but only use it to exchange packets with a fixed communication partner ("device-to-cloud" scenarios, see also Section 2.2 of [RFC7452]). More general devices are prepared to communicate with other nodes in the Internet as well.

We define the following classes of Internet technology level:

Name	Internet technology
I0	none (local interconnect only)
I1	device-to-cloud only
I9	full Internet connectivity supported

Table 11

5.3. Classes of physical layer bit rate

[This section is a trial balloon. We could also talk about burst rate, sustained rate; bits/s, messages/s, ...]

Physical layer technologies used by constrained devices can be categorized on the basis of physical layer (PHY) bit rate. The PHY bit rate class of a technology has important implications with regard to compatibility with existing protocols and mechanisms on the Internet, responsiveness to frame transmissions and need for header compression techniques.

We define the following classes of PHY bit rate:

Name	PHY bit rate (bit/s)	Comment
B0	< 10	Transmission time of 150-byte frame > MSL
B1	10 -- 10^3	Unresponsiveness if human expects reaction to sent frame (frame size > 62.5 byte)
B2	10^3 -- 10^6	Responsiveness if human expects reaction to sent frame, but header compression still needed
B3	> 10^6	Header compression yields relatively low performance benefits

Table 12

(note: 'Bx' stands for 'Bit rate x')

B0 technologies lead to very high transmission times, which may be close to or even greater than the Maximum Segment Lifetime (MSL) assumed on the Internet [RFC0793]. Many Internet protocols and mechanisms will fail when transmit times are greater than the MSL. B0 technologies lead to a frame transmission time greater than the MSL for a frame size greater than 150 bytes.

B1 technologies offer transmission times which are lower than the MSL (for a frame size greater than 150 bytes). However, transmission times for B1 technologies are still significant if a human expects a reaction to the transmission of a frame. With B1 technologies, the transmission time of a frame greater than 62.5 bytes exceeds 0.5 seconds, i.e. a threshold time beyond which any response or reaction to a frame transmission will appear not to be immediate [RFC5826].

B2 technologies do not incur responsiveness problems, but still benefit from using header compression techniques (e.g. [RFC6282]) to achieve performance improvements.

Over B3 technologies, the relative performance benefits of header compression are low. For example, in a duty-cycled technology offering B3 PHY bit rates, energy consumption decrease due to header compression may be comparable with the energy consumed while in a sleep interval. On the other hand, for B3 PHY bit rates, a human user will not be able to perceive whether header compression has been used or not in a frame transmission.

6. IANA Considerations

This document makes no requests to IANA.

7. Security Considerations

This document introduces common terminology that does not raise any new security issues. Security considerations arising from the constraints discussed in this document need to be discussed in the context of specific protocols. For instance, Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications of specific constraints on the security mechanisms employed. [RFC7416] provides a security threat analysis for the RPL routing protocol. Implementation considerations for security protocols on constrained nodes are discussed in [RFC7815] and [I-D.ietf-lwig-tls-minimal]. A wider view of security in constrained-node networks is provided in [RFC8576].

8. Informative References

- [FALL] Fall, K., "A Delay-Tolerant Network Architecture for Challenged Internets", SIGCOMM 2003, DOI 10.1145/863955.863960, 2003, <<https://doi.org/10.1145/863955.863960>>.
- [I-D.hui-vasseur-roll-rpl-deployment] Vasseur, J., Hui, J., Dasgupta, S., and G. Yoon, "RPL deployment experience in large scale networks", Work in Progress, Internet-Draft, draft-hui-vasseur-roll-rpl-deployment-01, 5 July 2012, <<https://www.ietf.org/archive/id/draft-hui-vasseur-roll-rpl-deployment-01.txt>>.
- [I-D.ietf-cbor-time-tag] Bormann, C., Gamari, B., and H. Birkholz, "Concise Binary Object Representation (CBOR) Tags for Time, Duration, and Period", Work in Progress, Internet-Draft, draft-ietf-cbor-time-tag-00, 19 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-cbor-time-tag-00.txt>>.

- [I-D.ietf-lwig-tls-minimal]
Kumar, S. S., Keoh, S. L., and H. Tschofenig, "A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks", Work in Progress, Internet-Draft, draft-ietf-lwig-tls-minimal-01, 7 March 2014, <<https://www.ietf.org/archive/id/draft-ietf-lwig-tls-minimal-01.txt>>.
- [IoT-2025] Rosen, M. and IDC, "Driving the Digital Agenda Requires Strategic Architecture", 16 November 2016, <https://idc-cema.com/dwn/SF_177701/driving_the_digital_agenda_requires_strategic_architecture_rosen_idc.pdf>. Slide 11
- [ISQ-13] International Electrotechnical Commission, "International Standard -- Quantities and units -- Part 13: Information science and technology", IEC 80000-13, March 2008.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [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>>.
- [RFC5826] Brandt, A., Buron, J., and G. Porcu, "Home Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5826, DOI 10.17487/RFC5826, April 2010, <<https://www.rfc-editor.org/info/rfc5826>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.

- [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>>.
- [RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <<https://www.rfc-editor.org/info/rfc6551>>.
- [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>>.
- [RFC7102] Vasseur, JP., "Terms Used in Routing for Low-Power and Lossy Networks", RFC 7102, DOI 10.17487/RFC7102, January 2014, <<https://www.rfc-editor.org/info/rfc7102>>.
- [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>>.
- [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>>.
- [RFC7416] Tsao, T., Alexander, R., Dohler, M., Daza, V., Lozano, A., and M. Richardson, Ed., "A Security Threat Analysis for the Routing Protocol for Low-Power and Lossy Networks (RPLs)", RFC 7416, DOI 10.17487/RFC7416, January 2015, <<https://www.rfc-editor.org/info/rfc7416>>.
- [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>>.

- [RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <<https://www.rfc-editor.org/info/rfc7452>>.
- [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>>.
- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [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>>.
- [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>>.
- [RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <<https://www.rfc-editor.org/info/rfc8576>>.
- [RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <<https://www.rfc-editor.org/info/rfc8724>>.
- [RFC9159] Gomez, C., Darroudi, S.M., Savolainen, T., and M. Spoerk, "IPv6 Mesh over BLUETOOTH(R) Low Energy Using the Internet Protocol Support Profile (IPSP)", RFC 9159, DOI 10.17487/RFC9159, December 2021, <<https://www.rfc-editor.org/info/rfc9159>>.
- [W432] "Warning about 4/32 devices", OpenWRT wiki, last accessed 2021-12-01, <https://openwrt.org/supported_devices/432_warning>.

[WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", Wiley-Blackwell monograph, DOI 10.1002/9780470686218, ISBN 9780470747995, 2009, <<https://doi.org/10.1002/9780470686218>>.

Acknowledgements

TBD

Authors' Addresses

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Mehmet Ersue
Munich
Germany
Email: mersue@gmail.com

Ari Keranen
Ericsson
Hirsalantie 11
FI-02420 Jorvas
Finland
Email: ari.keranen@ericsson.com

Carles Gomez
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Phone: +34-93-413-7206
Email: carlesgo@entel.upc.edu

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2020

C. Bormann
Universitaet Bremen TZI
T. Watteyne
Analog Devices
March 09, 2020

Virtual reassembly buffers in 6LoWPAN
draft-ietf-lwig-6lowpan-virtual-reassembly-02

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 September 10, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
2. Reassembly buffers	3
3. Virtual reassembly	3
4. Header compression	4
5. IANA Considerations	4
6. Security considerations	4
7. References	5
7.1. Normative References	5
7.2. Informative References	5
Acknowledgements	5
Authors' Addresses	5

1. Introduction

6LoWPAN [RFC4944] is the seminal standard for the transmission of IPv6 packets over IEEE 802.15.4 networks and has served as a blueprint for a number of related standards addressing low-power radios and other IoT connectivity solutions (the "6Lo suite").

One of the problems that need to be solved to enable sending IPv6 packets over low-power radios is that some of these (including IEEE 802.15.4) do not support frames that are large enough to hold IPv6 packets of the minimum MTU (Maximum Transmission Unit) defined for IPv6, 1280 bytes. This necessitates providing a fragmentation or segmentation scheme in the IP adaptation layer for the radio.

When employing adaptation layer fragmentation on constrained-node networks [RFC7228], 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 [BOOK], this has not been extensively described in the literature. The present document attempts to fill that gap.

[I-D.ietf-6lo-minimal-fragment] provides additional context and discussion about handling fragment forwarding in the 6Lo standards suite.

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:

- o datagram_size,
- o datagram_tag and L2 sender and receiver addresses (to which the datagram_tag is local),
- o 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,
- o 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):

- o L2 destination address used for forwarding,
- o 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 length that stays fixed during forwarding. 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

There are many security considerations with using fragmentation in the first place, even with adaptation layer fragmentation (which is not accessible outside the range of that adaptation layer). Some of the more specific ones are documented in [I-D.ietf-6lo-minimal-fragment] and will not be duplicated here.

In general, sending on fragments early from a node will relieve the node that is doing the forwarding, but put additional onus on the next node. This may or may not be in favor of an attacker.

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.
- [I-D.ietf-6lo-minimal-fragment] Watteyne, T., Thubert, P., and C. Bormann, "On Forwarding 6LoWPAN Fragments over a Multihop IPv6 Network", draft-ietf-6lo-minimal-fragment-14 (work in progress), March 2020.
- [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>>.
- [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>>.

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.

Authors' Addresses

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

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

Thomas Watteyne
Analog Devices
32990 Alvarado-Niles Road, Suite 910
Union City, CA 94587
USA

Email: thomas.watteyne@analog.com

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

M. Kovatsch
ETH Zurich
O. Bergmann
C. Bormann, Ed.
Universitaet Bremen TZI
July 02, 2018

CoAP Implementation Guidance
draft-ietf-lwig-coap-06

Abstract

The Constrained Application Protocol (CoAP) is designed for resource-constrained nodes and networks such as sensor nodes in a low-power lossy network (LLN). Yet to implement this Internet protocol on Class 1 devices (as per RFC 7228, ~ 10 KiB of RAM and ~ 100 KiB of ROM) also lightweight implementation techniques are necessary. This document provides lessons learned from implementing CoAP for tiny, battery-operated networked embedded systems. In particular, it provides guidance on correct implementation of the CoAP specification RFC 7252, memory optimizations, and customized protocol parameters.

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 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Protocol Implementation	4
2.1. Client/Server Model	4
2.2. Message Processing	5
2.2.1. On-the-fly Processing	5
2.2.2. Internal Data Structure	6
2.3. Message ID Usage	7
2.3.1. Duplicate Rejection	7
2.3.2. MID Namespaces	8
2.3.3. Relaxation on the Server	8
2.3.4. Relaxation on the Client	9
2.4. Token Usage	10
2.4.1. Tokens for Observe	11
2.4.2. Tokens for Blockwise Transfers	12
2.5. Transmission States	12
2.5.1. Request/Response Layer	12
2.5.2. Message Layer	13
2.6. Out-of-band Information	14
2.7. Programming Model	15
2.7.1. Client	16
2.7.2. Server	16
3. Optimizations	17
3.1. Message Buffers	17
3.2. Retransmissions	18
3.3. Observable Resources	18
3.4. Blockwise Transfers	19
3.4.1. Generic Proxying of Block Messages	19
3.4.2. Atomic Blockwise Operations	20
3.5. Deduplication with Sequential MIDs	20
4. Alternative Configurations	23
4.1. Transmission Parameters	23
4.2. CoAP over IPv4	24
5. Binding to specific lower-layer APIs	24
5.1. Berkeley Socket Interface	24
5.1.1. Responding from the right address	24
5.1.2. Handling ICMP errors	25
5.2. Java	25
5.3. Multicast detection	26
5.4. DTLS	26

6.	CoAP on various transports	26
6.1.	CoAP over reliable transports	27
6.2.	Translating between transports	27
6.2.1.	Transport translation by proxies	27
6.2.2.	One-to-one Transport translation	28
7.	IANA considerations	28
8.	Security considerations	28
9.	Acknowledgements	28
10.	References	28
10.1.	Normative References	28
10.2.	Informative References	29
	Authors' Addresses	30

1. Introduction

The Constrained Application Protocol [RFC7252] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios therefore include building automation, process optimization, and the Internet of Things. The major design objectives have been set on small protocol overhead, robustness against packet loss, and against high latency induced by small bandwidth shares or slow request processing in end nodes. To leverage integration of constrained nodes with the world-wide Internet, the protocol design was led by the REST architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [RFC7230].

Lightweight implementations benefit from this design in many respects: First, the use of Uniform Resource Identifiers (URIs) for naming resources and the transparent forwarding of their representations in a server-stateless request/response protocol make protocol translation to HTTP a straightforward task. Second, the set of protocol elements that are unavoidable for the core protocol, and thus must be implemented on every node, has been kept very small, minimizing the unnecessary accumulation of "optional" features. Options that - when present - are critical for message processing are explicitly marked as such to force immediate rejection of messages with unknown critical options. Third, the syntax of protocol data units is easy to parse and is carefully defined to avoid creation of state in servers where possible.

Although these features enable lightweight implementations of the Constrained Application Protocol, there is still a tradeoff between robustness and latency of constrained nodes on one hand and resource demands on the other. For constrained nodes of Class 1 or even Class 2 [RFC7228], the most limiting factors usually are dynamic memory needs, static code size, and energy. Most implementations

therefore need to optimize internal buffer usage, omit idle protocol features, and maximize sleeping cycles.

The present document gives possible strategies to solve this tradeoff for very constrained nodes (i.e., Class 1). For this, it provides guidance on correct implementation of the CoAP specification [RFC7252], memory optimizations, and customized protocol parameters.

2. Protocol Implementation

In the programming styles supported by very simple operating systems as found on constrained nodes, preemptive multi-threading is not an option. Instead, all operations are triggered by an event loop system, e.g., in a send-receive-dispatch cycle. It is also common practice to allocate memory statically to ensure stable behavior, as no memory management unit (MMU) or other abstractions are available. For a CoAP node, the two key parameters for memory usage are the number of (re)transmission buffers and the maximum message size that must be supported by each buffer. Often the maximum message size is set far below the 1280-byte MTU of 6LoWPAN to allow more than one open Confirmable transmission at a time (in particular for parallel observe notifications [RFC7641]). Note that implementations on constrained platforms often not even support the full MTU. Larger messages must then use blockwise transfers [RFC7959], while a good tradeoff between 6LoWPAN fragmentation and CoAP header overhead must be found. Usually the amount of available free RAM dominates this decision. For Class 1 devices, the maximum message size is typically 128 or 256 bytes (blockwise) payload plus an estimate of the maximum header size for the worst case option setting.

2.1. Client/Server Model

In general, CoAP servers can be implemented more efficiently than clients. REST allows them to keep the communication stateless and piggy-backed responses are not stored for retransmission, saving buffer space. The use of idempotent requests also allows to relax deduplication, which further decreases memory usage. It is also easy to estimate the required maximum size of message buffers, since URI paths, supported options, and maximum payload sizes of the application are known at compile time. Hence, when the application is distributed over constrained and unconstrained nodes, the constrained ones should preferably have the server role.

HTTP-based applications have established an inverse model because of the need for simple push notifications: A constrained client uses POST requests to update resources on an unconstrained server whenever an event (e.g., a new sensor reading) is triggered. This requirement is solved by the Observe option [RFC7641] of CoAP. It allows servers

to initiate communication and send push notifications to interested client nodes. This allows a more efficient and also more natural model for CoAP-based applications, where the information source is an origin server, which can also benefit from caching. Publish-subscribe brokers [I-D.ietf-core-coap-pubsub] may be deployed to act in the server role on behalf of constrained clients.

2.2. Message Processing

Apart from the required buffers, message processing is symmetric for clients and servers. First the base header has to be parsed and thereby checked if it is a valid CoAP message. For UDP datagrams, the version identifier or a size smaller than four bytes identify non-CoAP data. These datagrams need to be silently ignored. Other message format errors, such as an incomplete datagram or the usage of reserved values, may need to be rejected with a Reset (RST) message (see Section 4.2 and 4.3 of [RFC7252] for details).

As CoAP over TCP has a different base header, the Length field must be parsed to determine the message size. As this field may have up to five bytes, it may be extended over TCP segment boundaries. For CoAP over WebSockets the actual message length is given by the WebSocket frame hence the Length field is always zero.

Next, the token length is read based on the TKL field which is for all transports contained in the four least significant bits of the first byte. The (possibly empty) Token itself is located immediately after the four-byte base header for UDP, while for TCP and WebSockets, it follows the variable Length field and Code byte.

For the options following the Token, there are two alternatives: either process them on the fly when an option is accessed or initially parse all values into an internal data structure.

2.2.1. On-the-fly Processing

The advantage of on-the-fly processing is that no additional memory needs to be allocated to store the option values, which are stored efficiently inline in the buffer for incoming messages. Once the message is accepted for further processing, the set of options contained in the received message must be decoded to check for unknown critical options. To avoid multiple passes through the option list, the option parser might maintain a bit-vector where each bit represents an option number that is present in the received request. With the wide and sparse range of option numbers, the number itself cannot be used to indicate the number of left-shift operations to mask the corresponding bit. Hence, an implementation-specific enum of supported options should be used to mask the present

options of a message in the bitmap. In addition, the byte index of every option (a direct pointer) can be added to a sparse list (e.g., a one-dimensional array) for fast retrieval.

This particularly enables efficient handling of options that might occur more than once such as Uri-Path. In this implementation strategy, the delta is zero for any subsequent path segment, hence the stored byte index for this option (e.g., 11 for Uri-Path) would be overwritten to hold a pointer to only the last occurrence of that option. The Uri-Path can be resolved on the fly, though, and a pointer to the targeted resource stored directly in the sparse list.

Once the option list has been processed, all known critical option and all elective options can be masked out in the bit-vector to determine if any unknown critical option was present. If this is the case, this information can be used to create a 4.02 response accordingly. Note that full processing must only be done up to the highest supported option number. Beyond that, only the least significant bit (Critical or Elective) needs to be checked. Otherwise, if all critical options are supported, the sparse list of option pointers is used for further handling of the message.

2.2.2. Internal Data Structure

Using an internal data structure for all parsed options has an advantage when working on the option values, as they are already in a variable of corresponding type (e.g., an integer in host byte order). The incoming payload and byte strings of the header can be accessed directly in the buffer for incoming messages using pointers (similar to on-the-fly processing). This approach also benefits from a bitmap. Otherwise special values must be reserved to encode an unset option, which might require a larger type than required for the actual value range (e.g., a 32-bit integer instead of 16-bit).

Many of the byte strings (e.g., the URI) are usually not required when generating the response. When all important values are copied (e.g., the Token, which needs to be mirrored), the internal data structure facilitates using the buffer for incoming messages also for the assembly of outgoing messages - which can be the shared IP buffer provided by the operating system.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number and do not need to care about the order for the delta encoding. The CoAP encoding is applied in a serialization step before sending. In contrast, assembling outgoing messages with on-the-fly processing requires either extensive memmove operations to

insert new options, or restrictions for developers to set options in their correct order.

2.3. Message ID Usage

Many applications of CoAP use unreliable transports, in particular UDP, which can lose, reorder, and duplicate messages. Although DTLS's replay protection deals with duplication by the network, losses are addressed with DTLS retransmissions only for the handshake protocol and not for the application data protocol. Furthermore, CoAP implementations usually send CON retransmissions in new DTLS records, which are not considered duplicates at the DTLS layer.

2.3.1. Duplicate Rejection

CoAP's messaging sub-layer has been designed with protocol functionality such that rejection of duplicate messages is always possible. It is realized through the Message IDs (MIDs) and their lifetimes with regard to the message type.

Duplicate detection is under the discretion of the recipient (see Section 4.5 of [RFC7252], Section 2.3.3, Section 2.3.4). Where it is desired, the receiver needs to keep track of MIDs to filter the duplicates for at least `NON_LIFETIME` (145 s). This time also holds for CON messages, since it equals the possible reception window of `MAX_TRANSMIT_SPAN` + `MAX_LATENCY`.

On the sender side, MIDs of CON messages must not be re-used within the `EXCHANGE_LIFETIME`; MIDs of NONs respectively within the `NON_LIFETIME`. In typical scenarios, however, senders will re-use MIDs with intervals far larger than these lifetimes: with sequential assignment of MIDs, coming close to them would require 250 messages per second, much more than the bandwidth of constrained networks would usually allow for.

In cases where senders might come closer to the maximum message rate, it is recommended to use more conservative timings for the re-use of MIDs. Otherwise, opposite inaccuracies in the clocks of sender and recipient may lead to obscure message loss. If needed, higher rates can be achieved by using multiple endpoints for sending requests and managing the local MID per remote endpoint instead of a single counter per system (essentially extending the 16-bit message ID by a 16-bit port number and/or an 128-bit IP address). In controlled scenarios, such as real-time applications over industrial Ethernet, the protocol parameters can also be tweaked to achieve higher message rates (Section 4.1).

2.3.2. MID Namespaces

MIDs are assigned under the control of the originator of CON and NON messages, and they do not mix with the MIDs assigned by the peer for CON and NON in the opposite direction. Hence, CoAP implementors need to make sure to manage different namespaces for the MIDs used for deduplication. MIDs of outgoing CONs and NONs belong to the local endpoint; so do the MIDs of incoming ACKs and RSTs. Accordingly, MIDs of incoming CONs and NONs and outgoing ACKs and RSTs belong to the corresponding remote endpoint. Figure 1 depicts a scenario where mixing the namespaces would cause erroneous filtering.

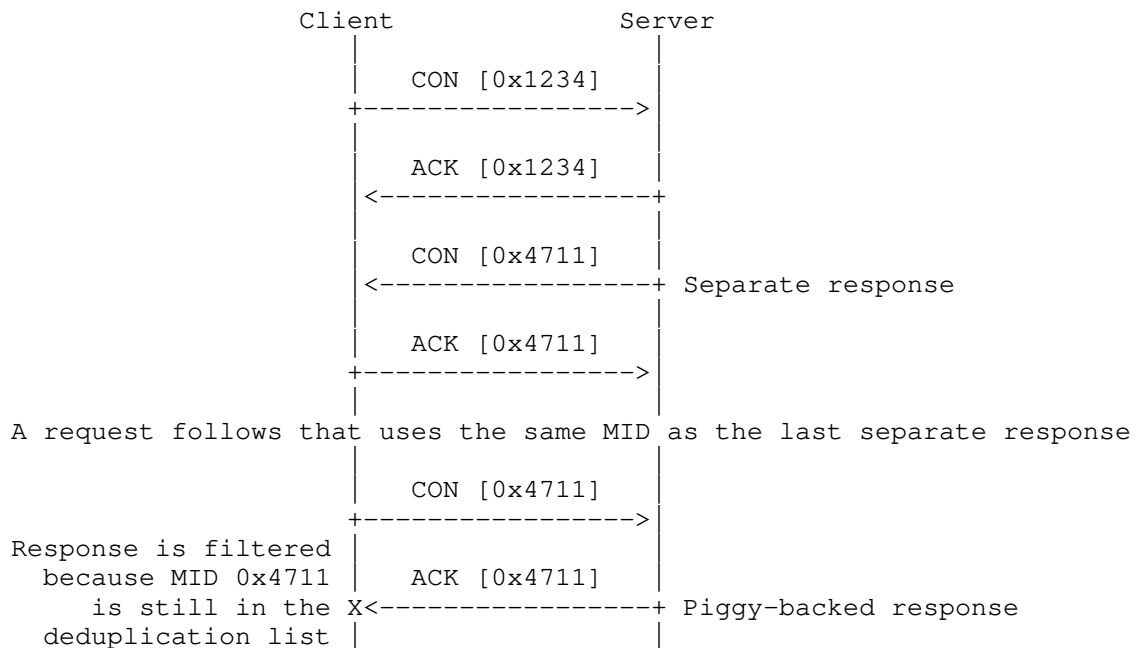


Figure 1: Deduplication must manage the MIDs in different namespace corresponding to their origin endpoints.

2.3.3. Relaxation on the Server

Using the de-duplication functionality is at the discretion of the receiver: Processing of duplicate messages comes at a cost, but so does the management of the state associated with duplicate rejection. The number of remote endpoints that need to be managed might be vast. This can be costly in particular for less constrained nodes that have throughput in the order of hundreds of thousands requests per second (which needs about 16 GiB of RAM just for duplicate rejection). Deduplication is also heavy for servers on Class 1 devices, as also

piggy-backed responses need to be stored for the case that the ACK message is lost. Hence, a receiver may have good reasons to decide not to perform deduplication. This behavior is possible when the application is designed with idempotent operations only and makes good use of the If-Match/If-None-Match options.

If duplicate rejection is indeed necessary (e.g., for non-idempotent requests) it is important to control the amount of state that needs to be stored. It can be reduced, for instance, by deduplication at resource level: Knowledge of the application and supported representations can minimize the amount of state that needs to be kept.

2.3.4. Relaxation on the Client

Duplicate rejection on the client side can be simplified by choosing clever Tokens that are virtually not re-used (e.g., through an obfuscated sequence number in the Token value) and only filter based on the list of open Tokens. If a client wants to re-use Tokens (e.g., the empty Token for optimizations), it requires strict duplicate rejection based on MIDs to avoid the scenario outlined in Figure 2.

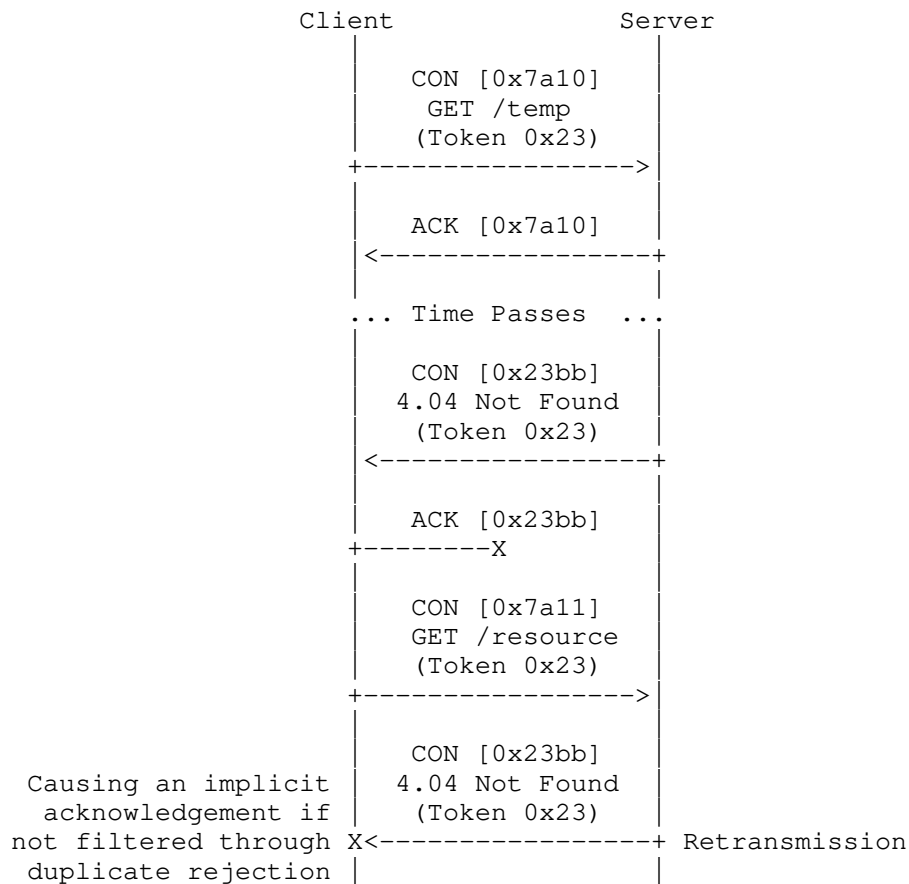


Figure 2: Re-using Tokens requires strict duplicate rejection.

2.4. Token Usage

Tokens are chosen by the client and help to identify request/response pairs that span several message exchanges (e.g., a separate response, which has a new MID). Servers do not generate Tokens and only mirror what they receive from the clients. Tokens must be unique within the namespace of a client throughout their lifetime. This begins when being assigned to a request and ends when the open request is closed by receiving and matching the final response. Neither empty ACKs nor notifications (i.e., responses carrying the Observe option) terminate the lifetime of a Token.

As already mentioned, a clever assignment of Tokens can help to simplify duplicate rejection. Yet this is also important for coping with client crashes. When a client restarts during an open request

and (unknowingly) re-uses the same Token, it might match the response from the previous request to the current one. Hence, when only the Token is used for matching, which is always the case for separate responses, randomized Tokens with enough entropy should be used. The 8-byte range for Tokens can even allow for one-time usage throughout the lifetime of a client node. When DTLS is used, client crashes/restarts will lead to a new security handshake, thereby solving the problem of mismatching responses and/or notifications.

2.4.1. Tokens for Observe

In the case of Observe [RFC7641], a request will be answered with multiple notifications and it is important to continue keeping track of the Token that was used for the request - its lifetime will end much later. Upon establishing an Observe relationship, the Token is registered at the server. Hence, the client's use of that specific Token is now limited to controlling the Observation relationship. A client can use it to cancel the relationship, which frees the Token upon success (i.e., the message with an Observe Option with the value set to 'deregister' (1) is confirmed with a response; see [RFC7641] section 3.6). However, the client might never receive the response due to a temporary network outage or worse, a server crash. Although a network outage will also affect notifications so that the Observe garbage collection could apply, the server might simply happen not to send CON notifications during that time. Alternative Observe lifetime models such as Stubbornness(tm) might also keep relationships alive for longer periods.

Thus, it is best to carefully choose the Token value used with Observe requests. (The empty value will rarely be applicable.) One option is to assign and re-use dedicated Tokens for each Observe relationship the client will establish. The choice of Token values also is critical in NoSec mode, to limit the effectiveness of spoofing attacks. Here, the recommendation is to use randomized Tokens with a length of at least four bytes (see Section 5.3.1 of [RFC7252]). Thus, dedicated ranges within the 8-byte Token space should be used when in NoSec mode. This also solves the problem of mismatching notifications after a client crash/restart.

When the client wishes to reinforce its interest in a resource, maybe not really being sure whether the server has forgotten it or not, the Token value allocated to the Observe relationship is used to re-register that observation (see Section 3.3.1 of [RFC7641] for details): If the server is still aware of the relationship (an entry with a matching endpoint and token is already present in its list of observers for the resource), it will not add a new relationship but will replace or update the existing one (Section 4.1 of [RFC7641]).

If not, it will simply establish a new registration which of course also uses the Token value.

If the client sends an Observe request for the same resource with a new Token, this is not a protocol violation, because the specification allows the client to observe the same resource in a different Observe relationship if the cache-key is different (e.g., requesting a different Content-Format). If the cache-key is not different, though, an additional Observe relationship just wastes the server's resources, and is therefore not allowed; the server might rely on this for its housekeeping.

2.4.2. Tokens for Blockwise Transfers

In general, blockwise transfers are independent from the Token and are correlated through client endpoint address and server address and resource path (destination URI). Thus, each block may be transferred using a different Token. Still it can be beneficial to use the same Token (it is freed upon reception of a response block) for all blocks, e.g., to easily route received blocks to the same response handler.

When Block2 is combined with Observe, notifications only carry the first block and it is up to the client to retrieve the remaining ones. These GET requests do not carry the Observe option and need to use a different Token, since the Token from the notification is still in use.

2.5. Transmission States

CoAP endpoints must keep transmission state to manage open requests, to handle the different response modes, and to implement reliable delivery at the message layer. The following finite state machines (FSMs) model the transmissions of a CoAP exchange at the request/response layer and the message layer. These layers are linked through actions. The `M_CMD()` action triggers a corresponding transition at the message layer and the `RR_EVT()` action triggers a transition at the request/response layer. The FSMs also use guard conditions to distinguish between information that is only available through the other layer (e.g., whether a request was sent using a CON or NON message).

2.5.1. Request/Response Layer

Figure 3 depicts the two states at the request/response layer of a CoAP client. When a request is issued, a "reliable_send" or "unreliable_send" is triggered at the message layer. The WAITING state can be left through three transitions: Either the client

cancels the request and triggers cancellation of a CON transmission at the message layer, the client receives a failure event from the message layer, or a receive event containing a response.

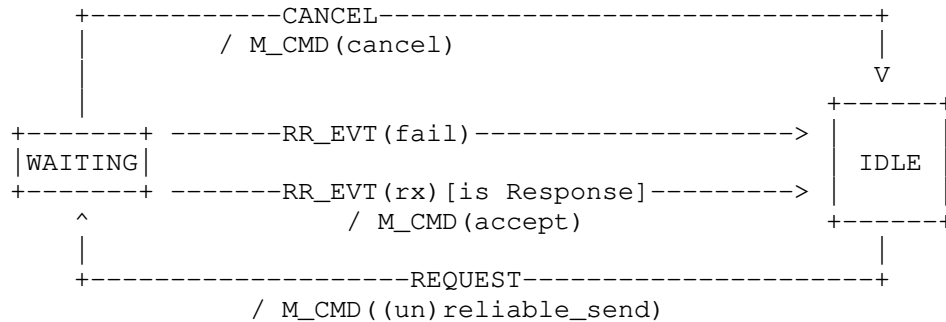


Figure 3: CoAP Client Request/Response Layer FSM

A server resource can decide at the request/response layer whether to respond with a piggy-backed or a separate response. Thus, there are two busy states in Figure 4, SERVING and SEPARATE. An incoming receive event with a NON request directly triggers the transition to the SEPARATE state.

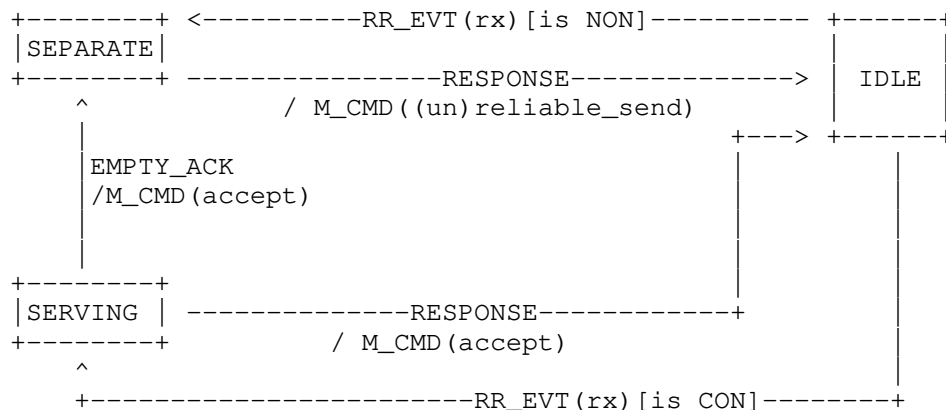
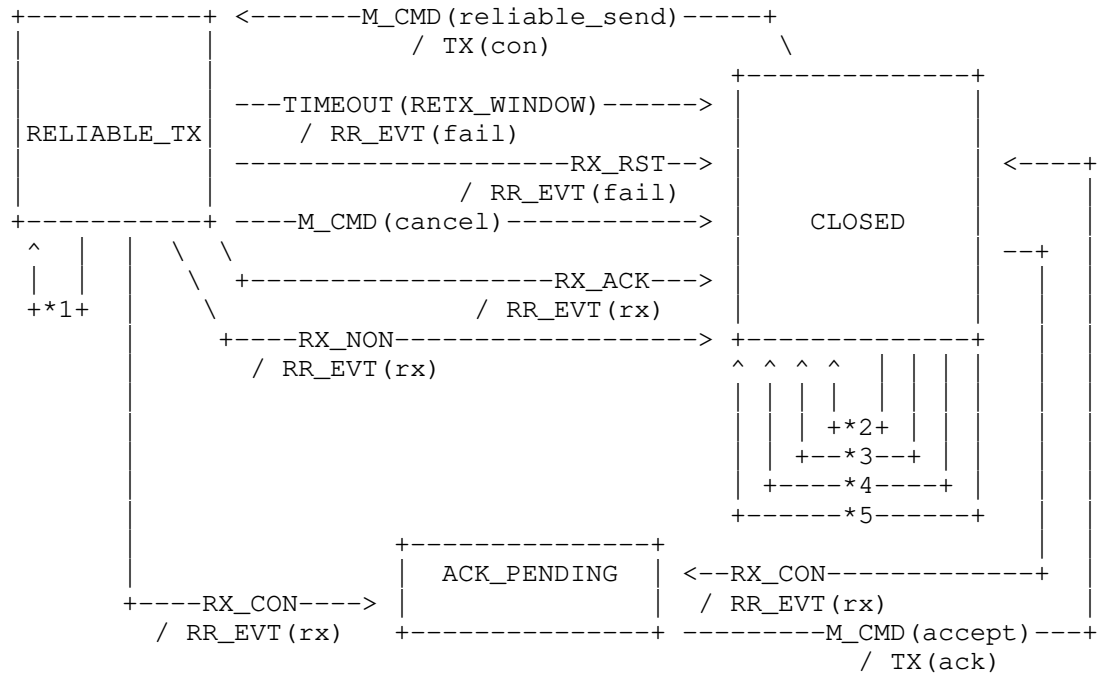


Figure 4: CoAP Server Request/Response Layer FSM

2.5.2. Message Layer

Figure 5 shows the different states of a CoAP endpoint per message exchange. Besides the linking action RR_EVT(), the message layer has a TX action to send a message. For sending and receiving NONs, the endpoint remains in its CLOSED state. When sending a CON, the endpoint remains in RELIABLE_TX and keeps retransmitting until the

transmission times out, it receives a matching RST, the request/response layer cancels the transmission, or the endpoint receives an implicit acknowledgement through a matching NON or CON. Whenever the endpoint receives a CON, it transitions into the ACK_PENDING state, which can be left by sending the corresponding ACK.



- *1: TIMEOUT(RETX_TIMEOUT) / TX(con)
- *2: M_CMD(unreliable_send) / TX(non)
- *3: RX_NON / RR_EVT(rx)
- *4: RX_RST / REMOVE_OBSERVER
- *5: RX_ACK

Figure 5: CoAP Message Layer FSM

T.B.D.: (i) Rejecting messages (can be triggered at message and request/response layer). (ii) ACKs can also be triggered at both layers.

2.6. Out-of-band Information

The CoAP implementation can also leverage out-of-band information, that might also trigger some of the transitions shown in Section 2.5. In particular ICMP messages can inform about unreachable remote endpoints or whole network outages. This information can be used to

pause or cancel ongoing transmission to conserve energy. Providing ICMP information to the CoAP implementation is easier in constrained environments, where developers usually can adapt the underlying OS (or firmware). This is not the case on general purpose platforms that have full-fledged OSES and make use of high-level programming frameworks.

The most important ICMP messages are host, network, port, or protocol unreachable errors. After appropriate vetting (cf. [RFC5927]), they should cause the cancellation of ongoing CON transmissions and clearing (or deferral) of Observe relationships. Requests to this destination should be paused for a sensible interval. In addition, the device could indicate of this error through a notification to a management endpoint or external status indicator, since the cause could be a misconfiguration or general unavailability of the required service. Problems reported through the Parameter Problem message are usually caused through a similar fundamental problem.

The CoAP specification recommends to ignore Source Quench and Time Exceeded ICMP messages, though. Source Quench messages were originally intended to inform the sender to reduce the rate of packets. However, this mechanism is deprecated through [RFC6633]. CoAP also comes with its own congestion control mechanism, which is already designed conservatively. One advanced mechanism that can be employed for better network utilization is CoCoA, [I-D.ietf-core-cocoa]. Time Exceeded messages often occur during transient routing loops (unless they are caused by a too small initial Hop Limit value).

2.7. Programming Model

The event-driven approach, which is common in event-loop-based firmware, has also proven very efficient for embedded operating systems [TinyOS], [Contiki]. Note that an OS is not necessarily required and a traditional firmware approach can suffice for Class 1 devices. Event-driven systems use split-phase operations (i.e., there are no blocking functions, but functions return and an event handler is called once a long-lasting operation completes) to enable cooperative multi-threading with a single stack.

Bringing a Web transfer protocol to constrained environments does not only change the networking of the corresponding systems, but also the programming model. The complexity of event-driven systems can be hidden through APIs that resemble classic RESTful Web service implementations.

2.7.1. Client

An API for asynchronous requests with response handler functions goes hand-in-hand with the event-driven approach. Synchronous requests with a blocking send function can facilitate applications that require strictly ordered, sequential request execution (e.g., to control a physical process) or other checkpointing (e.g., starting operation only after registration with the resource directory was successful). However, this can also be solved by triggering the next operation in the response handlers. Furthermore, as mentioned in Section 2.1, it is more like that complex control flow is done by more powerful devices and Class 1 devices predominantly run a CoAP server (which might include a minimal client to communicate with a resource directory).

2.7.2. Server

On CoAP servers, the event-driven nature can be hidden through resource handler abstractions as known from traditional REST frameworks. The following types of RESTful resources have proven useful to provide an intuitive API on constrained event-driven systems:

NORMAL A normal resource defined by a static Uri-Path and an associated resource handler function. Allowed methods could already be filtered by the implementation based on flags. This is the basis for all other resource types.

PARENT A parent resource manages several sub-resources under a given base path by programmatically evaluating the Uri-Path. Defining a URI template (see [RFC6570]) would be a convenient way to pre-parse arguments given in the Uri-Path.

PERIODIC A resource that has an additional handler function that is triggered periodically by the CoAP implementation with a resource-specific interval. It can be used to sample a sensor or perform similar periodic updates of its state. Usually, a periodic resource is observable and sends the notifications by triggering its normal resource handler from the periodic handler. These periodic tasks are quite common for sensor nodes, thus it makes sense to provide this functionality in the CoAP implementation and avoid redundant code in every resource.

EVENT An event resource is similar to an periodic resource, only that the second handler is called by an irregular event such as a button.

SEPARATE Separate responses are usually used when handling a request takes more time, e.g., due to a slow sensor or UART-based subsystems. To not fully block the system during this time, the handler should also employ split-phase execution: The resource handler returns as soon as possible and an event handler resumes responding when the result is ready. The separate resource type can abstract from the split-phase operation and take care of temporarily storing the request information that is required later in the result handler to send the response (e.g., source address and Token).

3. Optimizations

3.1. Message Buffers

The cooperative multi-threading of an event loop system allows to optimize memory usage through in-place processing and reuse of buffers, in particular the IP buffer provided by the OS or firmware.

CoAP servers can significantly benefit from in-place processing, as they can create responses directly in the incoming IP buffer. Note that an embedded OS usually only has a single buffer for incoming and outgoing IP packets. The first few bytes of the basic header are usually parsed into an internal data structure and can be overwritten without harm. Thus, empty ACKs and RST messages can promptly be assembled and sent using the IP buffer. Also when a CoAP server only sends piggy-backed or Non-confirmable responses, no additional buffer is required at the application layer. This, however, requires careful timing so that no incoming data is overwritten before it was processed. Because of cooperative multi-threading, this requirement is relaxed, though. Once the message is sent, the IP buffer can accept new messages again. This does not work for Confirmable messages, however. They need to be stored for retransmission and would block any further IP communication.

Depending on the number of requests that can be handled in parallel, an implementation might create a stub response filled with any option that has to be copied from the original request to the separate response, especially the Token option. The drawback of this technique is that the server must be prepared to receive retransmissions of the previous (Confirmable) request to which a new acknowledgement must be generated. If memory is an issue, a single buffer can be used for both tasks: Only the message type and code must be updated, changing the message id is optional. Once the resource representation is known, it is added as new payload at the end of the stub response. Acknowledgements still can be sent as described before as long as no additional options are required to describe the payload.

3.2. Retransmissions

CoAP's reliable transmissions require the before-mentioned retransmission buffers. Messages, such as the requests of a client, should be stored in serialized form. For servers, retransmissions apply for Confirmable separate responses and Confirmable notifications [RFC7641]. As separate responses stem from long-lasting resource handlers, the response should be stored for retransmission instead of re-dispatching a stored request (which would allow for updating the representation). For Confirmable notifications, please see Section 2.6, as simply storing the response can break the concept of eventual consistency.

String payloads such as JSON require a buffer to print to. By splitting the retransmission buffer into header and payload part, it can be reused. First to generate the payload and then storing the CoAP message by serializing into the same memory. Thus, providing a retransmission for any message type can save the need for a separate application buffer. This, however, requires an estimation about the maximum expected header size to split the buffer and a memmove to concatenate the two parts.

For platforms that disable clock tick interrupts in sleep states, the application must take into consideration the clock deviation that occurs during sleep (or ensure to remain in idle state until the message has been acknowledged or the maximum number of retransmissions is reached). Since CoAP allows up to four retransmissions with a binary exponential back-off it could take up to 45 seconds until the send operation is complete. Even in idle state, this means substantial energy consumption for low-power nodes. Implementers therefore might choose a two-step strategy: First, do one or two retransmissions and then, in the later phases of back-off, go to sleep until the next retransmission is due. In the meantime, the node could check for new messages including the acknowledgement for any Confirmable message to send.

3.3. Observable Resources

For each observer, the server needs to store at least address, port, token, and the last outgoing message ID. The latter is needed to match incoming RST messages and cancel the observe relationship.

It is favorable to have one retransmission buffer per observable resource that is shared among all observers. Each notification is serialized once into this buffer and only address, port, and token are changed when iterating over the observer list (note that different token lengths might require realignment). The advantage becomes clear for Confirmable notifications: Instead of one

retransmission buffer per observer, only one buffer and only individual retransmission counters and timers in the list entry need to be stored. When the notifications can be sent fast enough, even a single timer would suffice. Furthermore, per-resource buffers simplify the update with a new resource state during open deliveries.

3.4. Blockwise Transfers

Blockwise transfers have the main purpose of providing fragmentation at the application layer, where partial information can be processed. This is not possible at lower layers such as 6LoWPAN, as only assembled packets can be passed up the stack. While [RFC7959] also anticipates atomic handling of blocks, i.e., only fully received CoAP messages, this is not possible on Class 1 devices.

When receiving a blockwise transfer, each block is usually passed to a handler function that for instance performs stream processing or writes the blocks to external memory such as flash. Although there are no restrictions in [RFC7959], it is beneficial for Class 1 devices to only allow ordered transmission of blocks. Otherwise on-the-fly processing would not be possible.

When sending a blockwise transfer out of dynamically generated information, Class 1 devices usually do not have sufficient memory to print the full message into a buffer, and slice and send it in a second step. For instance, if the CoRE Link Format at `/.well-known/core` is dynamically generated, a generator function is required that generates slices of a large string with a specific offset length (a `'sonprintf()'`). This functionality is required recurrently and should be included in a library.

3.4.1. Generic Proxying of Block Messages

Proxies cannot ignore the Block options by specification, because the options Block1 and Block2 are not safe-to-forward. The rationale behind this design decision is that servers might not be able to distinguish blocks originating from different senders once they have been forwarded by a CoAP proxy. For atomic operations where all blocks are assembled before actually executing the desired operation, this could lead to inconsistent state on the server side.

To ensure that this does not happen, a proxy can add the Request-Tag option (see [I-D.ietf-core-echo-request-tag]) containing data that uniquely identifies the originating endpoint in the proxy namespace.

3.4.2. Atomic Blockwise Operations

When an implementation needs to assemble blocks from block-wise transfers, applications need to create an identifier to group messages that belong together. This "Block Key" at least contains:

- o The source endpoint (e.g., IP address and port in the UDP case),
- o the destination endpoint,
- o the Cache-Key (as updated in [RFC7252]), and
- o all options that are proxy unsafe and not explicitly described as safe for block-wise assembly.

The only known options safe for block-wise assembly are the options Block1 and Block2 [RFC7959].

For the Block1 phase, the request payload is excluded from the identifier generation as it is just being assembled.

If a message is received that is not the start of a block-wise operation has a Block Key that is not known, and the implementation needs to act atomically on a request body, it must answer 4.08 (Request Entity Incomplete).

Conversely, clients should be aware that requests whose Block Key matches can be interpreted by the server atomically. This especially affects proxies (see Section 3.4.1).

3.5. Deduplication with Sequential MIDs

CoAP's duplicate rejection functionality can be straightforwardly implemented in a CoAP endpoint by storing, for each remote CoAP endpoint ("peer") that it communicates with, a list of recently received CoAP Message IDs (MIDs) along with some timing information. A CoAP message from a peer with a MID that is in the list for that peer can simply be discarded.

The timing information in the list can then be used to time out entries that are older than the `_expected` extent of the re-ordering_, an upper bound for which can be estimated by adding the `_potential retransmission window_` ([RFC7252] section "Reliable Messages") and the time packets can stay alive in the network.

Such a straightforward implementation is suitable in case other CoAP endpoints generate random MIDs. However, this storage method may consume substantial RAM in specific cases, such as:

- o many clients are making periodic, non-idempotent requests to a single CoAP server;
- o one client makes periodic requests to a large number of CoAP servers and/or requests a large number of resources; where servers happen to mostly generate separate CoAP responses (not piggy-backed);

For example, consider the first case where the expected extent of re-ordering is 50 seconds, and N clients are sending periodic POST requests to a single CoAP server during a period of high system activity, each on average sending one client request per second. The server would need $100 * N$ bytes of RAM to store the MIDs only. This amount of RAM may be significant on a RAM-constrained platform. On a number of platforms, it may be easier to allocate some extra program memory (e.g. Flash or ROM) to the CoAP protocol handler process than to allocate extra RAM. Therefore, one may try to reduce RAM usage of a CoAP implementation at the cost of some additional program memory usage and implementation complexity.

Some CoAP clients generate MID values by using a Message ID variable [RFC7252] that is incremented by one each time a new MID needs to be generated. (After the maximum value 65535 it wraps back to 0.) We call this behavior "sequential" MIDs. One approach to reduce RAM use exploits the redundancy in sequential MIDs for a more efficient MID storage in CoAP servers.

Naturally such an approach requires, in order to actually reduce RAM usage in an implementation, that a large part of the peers follow the sequential MID behavior. To realize this optimization, the authors therefore RECOMMEND that CoAP endpoint implementers employ the "sequential MID" scheme if there are no reasons to prefer another scheme, such as randomly generated MID values.

Security considerations might call for a choice for (pseudo)randomized MIDs. Note however that with truly randomly generated MIDs the probability of MID collision is rather high in use cases as mentioned before, following from the Birthday Paradox. For example, in a sequence of 52 randomly drawn 16-bit values the probability of finding at least two identical values is about 2 percent.

From here on we consider efficient storage implementations for MIDs in CoAP endpoints, that are optimized to store "sequential" MIDs. Because CoAP messages may be lost or arrive out-of-order, a solution has to take into account that received MIDs of CoAP messages are not actually arriving in a sequential fashion, due to lost or reordered messages. Also a peer might reset and lose its MID counter(s) state.

In addition, a peer may have a single Message ID variable used in messages to many CoAP endpoints it communicates with, which partly breaks sequentiality from the receiving CoAP endpoint's perspective. Finally, some peers might use a randomly generated MID values approach. Due to these specific conditions, existing sliding window bitfield implementations for storing received sequence numbers are typically not directly suitable for efficiently storing MIDs.

Table 1 shows one example for a per-peer MID storage design: a table with a bitfield of a defined length $_K$ per entry to store received MIDs (one per bit) that have a value in the range $[MID_i + 1, MID_i + K]$.

MID base	K-bit bitfield	base time value
MID_0	010010101001	t_0
MID_1	111101110111	t_1
... etc.		

Table 1: A per-peer table for storing MIDs based on MID_i

The presence of a table row with base MID_i (regardless of the bitfield values) indicates that a value MID_i has been received at a time t_i. Subsequently, each bitfield bit k (0...K-1) in a row i corresponds to a received MID value of MID_i + k + 1. If a bit k is 0, it means a message with corresponding MID has not yet been received. A bit 1 indicates such a message has been received already at approximately time t_i. This storage structure allows e.g. with k=64 to store in best case up to 130 MID values using 20 bytes, as opposed to 260 bytes that would be needed for a non-sequential storage scheme.

The time values t_i are used for removing rows from the table after a preset timeout period, to keep the MID store small in size and enable these MIDs to be safely re-used in future communications. (Note that the table only stores one time value per row, which therefore needs to be updated on receipt of another MID that is stored as a single bit in this row. As a consequence of only storing one time value per row, older MID entries typically time out later than with a simple per-MID time value storage scheme. The endpoint therefore needs to ensure that this additional delay before MID entries are removed from the table is much smaller than the time period after which a peer starts to re-use MID values due to wrap-around of a peer's MID variable. One solution is to check that a value t_i in a table row

is still recent enough, before using the row and updating the value `t_i` to current time. If not recent enough, e.g. older than `N` seconds, a new row with an empty bitfield is created.) [Clearly, these optimizations would benefit if the peer were much more conservative about re-using MIDs than currently required in the protocol specification.]

The optimization described is less efficient for storing randomized MIDs that a CoAP endpoint may encounter from certain peers. To solve this, a storage algorithm may start in a simple MID storage mode, first assuming that the peer produces non-sequential MIDs. While storing MIDs, a heuristic is then applied based on monitoring some "hit rate", for example, the number of MIDs received that have a Most Significant Byte equal to that of the previous MID divided by the total number of MIDs received. If the hit rate tends towards 1 over a period of time, the MID store may decide that this particular CoAP endpoint uses sequential MIDs and in response improve efficiency by switching its mode to the bitfield based storage.

4. Alternative Configurations

4.1. Transmission Parameters

When a constrained network of CoAP nodes is not communicating over the Internet, for instance because it is shielded by a proxy or a closed deployment, alternative transmission parameters can be used. Consequently, the derived time values provided in [RFC7252] section 4.8.2 will also need to be adjusted, since most implementations will encode their absolute values.

Static adjustments require a fixed deployment with a constant number or upper bound for the number of nodes, number of hops, and expected concurrent transmissions. Furthermore, the stability of the wireless links should be evaluated. `ACK_TIMEOUT` should be chosen above the `xx%` percentile of the round-trip time distribution. `ACK_RANDOM_FACTOR` depends on the number of nodes on the network. `MAX_RETRANSMIT` should be chosen suitable for the targeted application. A lower bound for `LEISURE` can be calculated as

$$lb_Leisure = S * G / R$$

where `S` is the estimated response size, `G` the group size, and `R` the target data transfer rate (see [RFC7252] section 8.2). `NSTART` and `PROBING_RATE` depend on estimated network utilization. If the main cause for loss are weak links, higher values can be chosen.

Dynamic adjustments will be performed by advanced congestion control mechanisms such as [I-D.ietf-core-cocoa]. They are required if the

main cause for message loss is network or endpoint congestion. Semi-dynamic adjustments could be implemented by disseminating new static transmission parameters to all nodes when the network configuration changes (e.g., new nodes are added or long-lasting interference is detected).

4.2. CoAP over IPv4

CoAP was designed for the properties of IPv6, which is dominating in constrained environments because of the 6LoWPAN adaption layer [RFC6282]. In particular, the size limitations of CoAP are tailored to the minimal MTU of 1280 bytes. Until the transition towards IPv6 converges, CoAP nodes might also communicate over IPv4, though. Sections 4.2 and 4.6 of the base specification [RFC7252] already provide guidance and implementation notes to handle the smaller minimal MTUs of IPv4.

Another deployment issue in legacy IPv4 deployments is caused by Network Address Translators (NATs). The session timeouts are unpredictable and NATs may close UDP sessions with timeout as short as 60 seconds. This makes CoAP endpoints behind NATs practically unreachable, even when they contact the remote endpoint with a public IP address first. Incorrect behavior may also arise when the NAT session heuristic changes the external port between two successive CoAP messages. For the remote endpoint, this will look like two different CoAP endpoints on the same IP address. Such behavior can be fatal for the resource directory registration interface.

5. Binding to specific lower-layer APIs

Implementing CoAP on specific lower-layer APIs appears to consistently bring up certain less-known aspects of these APIs. This section is intended to alert implementers to such aspects.

5.1. Berkeley Socket Interface

5.1.1. Responding from the right address

In order for a client to recognize a reply (response or acknowledgement) as coming from the endpoint to which the initiating packet was addressed, the source IPv6 address of the reply needs to match the destination address of the initiating packet.

Implementers that have previously written TCP-based applications are used to binding their server sockets to `INADDR_ANY`. Any TCP connection received over such a socket is then more specifically bound to the source address from which the TCP connection setup was received; no programmer action is needed for this.

For stateless UDP sockets, more manual work is required. Simply receiving a packet from a UDP socket bound to `INADDR_ANY` loses the information about the destination address; replying to it through the same socket will use the default address established by the kernel. Two strategies are available:

- o Only use sockets bound to a specific address (not `INADDR_ANY`). A system with multiple interfaces (or addresses) will thus need to bind multiple sockets and send replies back on the same socket the initiating packet was received on.
- o Use `IPV6_RECVPKTINFO` [RFC3542] to configure the socket, and mirror back the `IPV6_PKTINFO` information for the reply (see also Section 5.1.1.1).

5.1.1.1. Managing interfaces

For some applications, it may further be relevant what interface is chosen to send to an endpoint, beyond the kernel choosing one that has a routing table entry for the destination address. E.g., it may be natural to send out a response or acknowledgment on the same interface that the packet prompting it was received. The end of the introduction to section 6 of [RFC3542] describes a simple technique for this, where that RFC's API (`IPV6_PKTINFO`) is available. The same data structure can be used for indicating an interface to send a packet that is initiating an exchange. (Choosing that interface is too application-specific to be in scope for the present document.)

5.1.2. Handling ICMP errors

Sockets that use the connect and send functions usually receive ICMP errors in the form of error codes, sockets that use `sendto` or `sendmsg` do not receive them at all.

Neither is sufficient to implement the guidance in Section 2.6, as the vetting of the message requires access to the CoAP headers in the ICMP error. The necessary information can be obtained by using the `IPV6_RECVERR` option.

5.2. Java

Java provides a wildcard address (`0.0.0.0`) to bind a socket to all network interface. This is useful when a server is supposed to listen on any available interface including the lookback address. For UDP, and hence CoAP this poses a problem, however, because the `DatagramPacket` class does not provide the information to which address it was sent. When replying through the wildcard socket, the JVM will pick the default address, which can break the correlation of

messages when the remote endpoint did not send the message to the default address. This is in particular precarious for IPv6 where it is common to have multiple IP addresses per network interface. Thus, it is recommended to bind to all addresses explicitly and manage the destination address of incoming messages within the CoAP implementation.

5.3. Multicast detection

Similar to the considerations above, Section 8 of [RFC7252] requires a node to detect whether a packet that it is going to reply to was sent to a unicast or to a multicast address. On most platforms, binding a UDP socket to a unicast address ensures that it only receives packets addressed to that address. Programmers relying on this property should ensure that it indeed applies to the platform they are using. If it does not, IPV6_PKTINFO may, again, help for Berkeley Socket Interfaces. For Java, explicit management of different sockets (in this case a MulticastSocket) is required.

5.4. DTLS

CoAPS implementations require access to the authenticated user/device principal to realize access control for resources. How this information can be accessed heavily depends on the DTLS implementation used. Generic and portable CoAP implementations might want to provide an abstraction layer that can be used by application developers that implement resource handlers. It is recommended to keep the API of such an application layer close to popular HTTPS solutions that are available for the targeted platform, for instance, `mod_ssl` or the Java Servlet API.

6. CoAP on various transports

As specified in [RFC7252], CoAP is defined for two underlying transports: UDP and DTLS. These transports are relatively similar in terms of the properties they expose to their users. (The main difference, apart from the increased security, is that DTLS provides an abstraction of a connection, into which the endpoint abstraction is placed; in contrast, the UDP endpoint abstraction is based on four-tuples of IP addresses and ports.)

Recently, the need to carry CoAP over other transports [I-D.silverajan-core-coap-alternative-transports] has led to specifications such as CoAP over TLS or TCP or WebSockets [RFC8323], or even over non-IP transports such as SMS [I-D.becker-core-coap-sms-gprs]. This section discusses considerations that arise when handling these different transports in an implementation.

6.1. CoAP over reliable transports

To cope with transports without reliable delivery (such as UDP and DTLS), CoAP defines its own message layer, with acknowledgments, timers, and retransmission. When CoAP is run over a transport that provides its own reliability (such as TCP or TLS), running this machinery would be redundant. Worse, keeping the machinery in place is likely to lead to interoperability problems as it is unlikely to be tested as well as on unreliable transports. Therefore, [I-D.silverajan-core-coap-alternative-transports] was defined by removing the message layer from CoAP and just running the request/response layer directly on top of the reliable transport. This also leads to a reduced (from the UDP/DTLS 4-byte header) header format.

Conversely, where reliable transports provide a byte stream abstraction, some form of message delimiting had to be added, which now needs to be handled in the CoAP implementation. The use of reliable transports may reduce the disincentive for using messages larger than optimal link layer packet sizes. Where different message sizes are chosen by an application for reliable and for unreliable transports, this can pose additional challenges for translators (Section 6.2).

Where existing CoAP APIs expose details of the the message layer (e.g., CON vs. NON, or assigning application layer semantics to ACKs), using a reliable transport may require additional adjustments.

6.2. Translating between transports

One obvious way to convey CoAP exchanges between different transports is to run a CoAP proxy that supports both transports. The usual considerations for proxies apply. Section 6.2.1 discusses some additional considerations.

Where not much of the functionality of CoAP proxies (such as caching) is required, a simpler 1:1 translation may be possible, as discussed in Section 6.2.2.

6.2.1. Transport translation by proxies

(TBD. In particular, point out the obvious: fan-in/fan-out means that separate message ID and token spaces need to be maintained at the ends of the proxy.)

One more CoAP specific function of a transport translator proxy may be to convert between different block sizes, e.g. between a TCP connection that can tolerate large blocks and UDP over a constrained node network.

6.2.2. One-to-one Transport translation

A translator with reduced requirements for state maintenance can be constructed when no fan-in or fan-out is required, and when the namespace lifetimes of the two sides can be made to coincide. For this one-to-one translation, there is no need to manage message-ID and Token value spaces for both sides separately. So, a simple UDP-to-UDP one-to-one translator could simply copy the messages (among other applications, this might be useful for translation between IPv4 and IPv6 spaces). Similarly, a DTLS-to-TCP translator could be built that executes the message layer (deduplication, retransmission) on the DTLS side, and repackages the CoAP header (add/remove the length information, and remove/add the message ID and message type) between the DTLS and the TCP side.

By definition, such a simple one-to-one translator needs to shut down the connection on one side when the connection on the other side terminates. However, a UDP-to-TCP one-to-one translator cannot simply shut down the UDP endpoint when the TCP endpoint vanishes because the TCP connection closes, so some additional management of state will be necessary.

7. IANA considerations

This document has no actions for IANA.

8. Security considerations

TBD

9. Acknowledgements

Esko Dijk contributed the sequential MID optimization. Xuan He provided help creating and improved the state machine charts. Christian Amsuess provided input on forwarding block messages by proxies and usage of the Request-Tag option.

10. References

10.1. Normative References

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,
"CoAP Simple Congestion Control/Advanced", draft-ietf-
core-cocoa-03 (work in progress), February 2018.

- [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>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <<https://www.rfc-editor.org/info/rfc6633>>.
- [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>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.

10.2. Informative References

- [Contiki] Dunkels, A., Groenvall, B., and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", Proceedings of the First IEEE Workshop on Embedded Networked Sensors, November 2004.
- [I-D.becker-core-coap-sms-gprs] Kuladinithi, K., Becker, M., Li, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-06 (work in progress), February 2017.

- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (work in progress), March 2018.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-02 (work in progress), June 2018.
- [I-D.silverajan-core-coap-alternative-transport]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-11 (work in progress), March 2018.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, DOI 10.17487/RFC3542, May 2003, <<https://www.rfc-editor.org/info/rfc3542>>.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <<https://www.rfc-editor.org/info/rfc5927>>.
- [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>>.
- [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>>.
- [TinyOS] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Woo, A., Hill, J., Welsh, M., Brewer, E., and D. Culler, "TinyOS: An Operating System for Sensor Networks", Ambient intelligence, Springer (Berlin Heidelberg), ISBN 978-3-540-27139-0, 2005.

Authors' Addresses

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
CH-8092 Zurich
Switzerland

Email: kovatsch@inf.ethz.ch

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Email: bergmann@tzi.org

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

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.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Requirements Language and Terminology	4
2. Neighbor Management	5
2.1. Significance of Neighbor management policy	5
2.2. Trivial neighbor management policies	6
2.3. Lifecycle of a NCE	7
2.3.1. NCE Insertion	7
2.3.2. NCE Deletion	10
2.3.3. NCE Eviction	11
2.3.3.1. Eviction for directly connected routing entries	11
2.3.4. NCE Reinforcement	12
2.4. Requirements of a good neighbor management policy	12
2.5. Approaches to neighbor management policy	13
2.5.1. Reactive Approach	13
2.5.2. Proactive Approach	13
3. Reservation based Neighbor Management Policy	14
3.1. Limitations of such a policy	16
4. Acknowledgements	16
5. IANA Considerations	17
6. Security Considerations	17
7. References	17
7.1. Normative References	17
7.2. Informative References	18
Appendix A. Performance Result	19
Authors' Addresses	20

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.

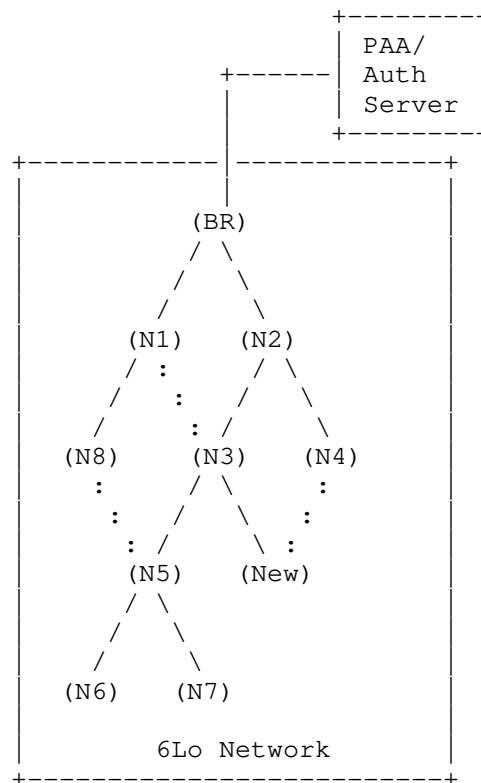


Figure 1: Sample Topology

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 than the neighbor cache size, the entries have to be prioritized. [Woo_et_al] and [Dawans_et_al] talk about prioritization of neighbor entries by using link quality estimation techniques. But prioritization alone may not necessarily be optimal in all cases. The reason or function why neighbor entry was added also needs to be taken in consideration. For example, evicting a routing direct child might have a ripple effect in turn impacting all the sub-children as well.

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

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

2.2. Trivial neighbor management policies

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

- a. First Come First Serve (FCFS) policy
- b. Least Recently Used (LRU) policy

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

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

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

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

2.3. Lifecycle of a NCE

2.3.1. NCE Insertion

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

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

Node Joining procedure: A new joinee node discovers a relay element to initiate its auth procedure. At the end of the discovery phase the new joinee node would have known the link local IP address of the relay element. The joinee node will send an unsecured-NS to the relay element to solicit its NA. The PRE may send a NA with the suitable status code as defined in section 6.5.3 of [RFC6775].

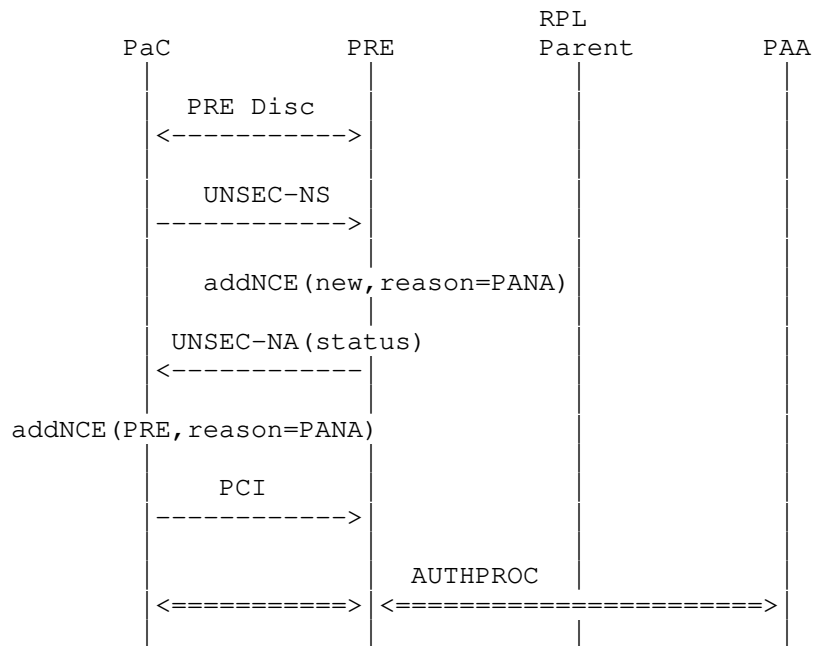


Figure 2: NCE creation between PaC and PRE during relay discovery process

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.

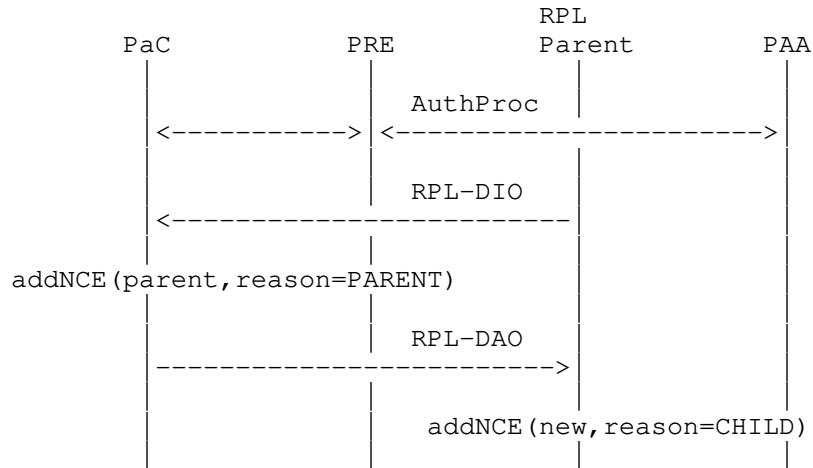


Figure 3: NCE creation call Flow for RPL storing MOP

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.

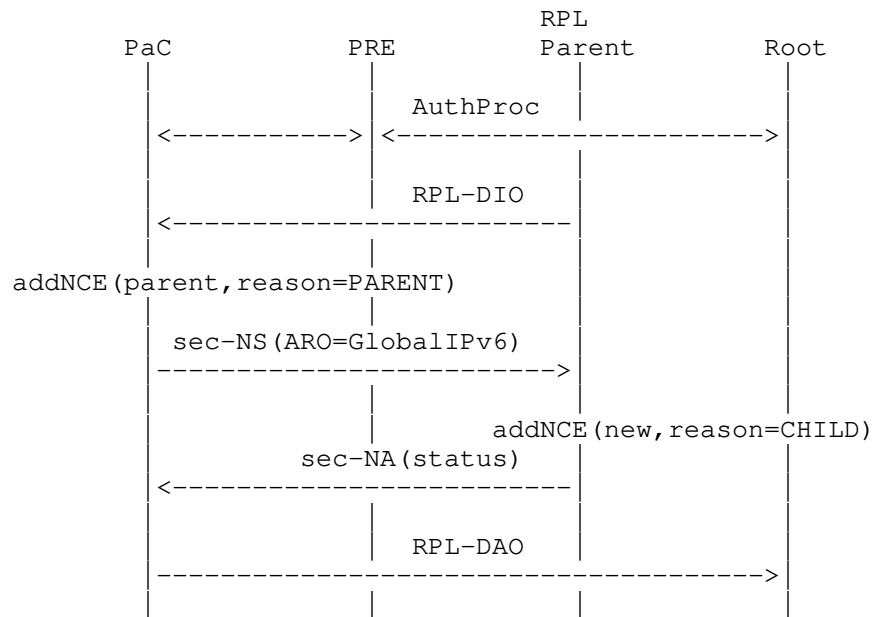


Figure 4: NCE creation call Flow for non-storing MOP

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 containing its link-local IPv6 address. NS helps to determine whether the PRE can allocate an NCE for the PaC. PRE accordingly sends a NA response with appropriate status field.

2.5.2. Proactive Approach

Neighbor cache availability could be proactively advertised by the parent nodes in the DIO messages and in the PRE discovery messages. A child RPL node may additionally use this information from DIO as part of parent selection process.

[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 depending 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.

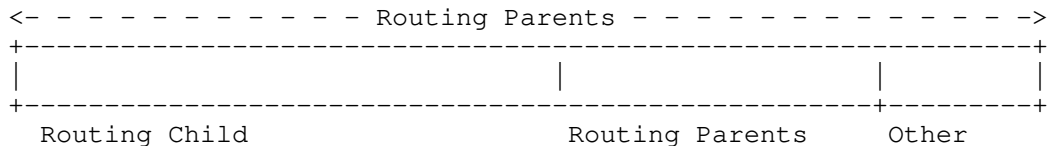


Figure 5: Reservation of NCEs in neighbor table

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 unreachabe 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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

J. Mattsson
F. Palombini
Ericsson AB
July 2, 2018

Comparison of CoAP Security Protocols
draft-ietf-lwig-security-protocol-comparison-01

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Overhead of Security Protocols	3
2.1. DTLS 1.2	3
2.1.1. DTLS 1.2	3
2.1.2. DTLS 1.2 with 6LoWPAN-GHC	4
2.1.3. DTLS 1.2 with Connection ID	4
2.1.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC	5
2.2. DTLS 1.3	6
2.2.1. DTLS 1.3	6
2.2.2. DTLS 1.3 with 6LoWPAN-GHC	6
2.2.3. DTLS 1.3 with Connection ID	7
2.2.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC	7
2.2.5. DTLS 1.3 with Short Header	8
2.2.6. DTLS 1.3 with Short Header and 6LoWPAN-GHC	8
2.3. TLS 1.2	9
2.3.1. TLS 1.2	9
2.3.2. TLS 1.2 with 6LoWPAN-GHC	9
2.4. TLS 1.3	10
2.4.1. TLS 1.3	10
2.4.2. TLS 1.3 with 6LoWPAN-GHC	10
2.5. OSCORE	11
3. Overhead with Different Parameters	12
4. Summary	14
5. Security Considerations	15
6. IANA Considerations	15
7. Informative References	15
Acknowledgments	16
Authors' Addresses	16

1. Introduction

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

2. Overhead of Security Protocols

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (e.g. AES_128_CCM_8 or AES-CCM-16-64), 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 DTLS 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 6LoWPAN-GHC [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 used 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 record layer 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 6LoWPAN-GHC [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 used 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 content type):

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 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when DTLS is used 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 content type):

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 content type):

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 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when DTLS is used 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 content type):

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

Short epoch and sequence:
30 05
Ciphertext (including encrypted content type):
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 6LoWPAN-GHC [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 content type):
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 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used 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 content type):

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 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used 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 content type):

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].

The below calculation Option Delta = '9', Sender ID = '' (empty string), and Sequence Number = '05', and is only an example. Note that Sender ID = '' (empty string) can only be used by one client per server.

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'
<hr/>			
DTLS 1.2	29	29	29
DTLS 1.3	16	16	16
DTLS 1.3 (short header)	11	11	11
<hr/>			
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	17	17	17
DTLS 1.3 (short header) (GCH)	12	12	12
<hr/>			
TLS 1.2	21	21	21
TLS 1.3	14	14	14
<hr/>			
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	15	16	17
<hr/>			
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'
<hr/>			
DTLS 1.2	29	30	31
DTLS 1.3	16	17	18
DTLS 1.3 (short header)	11	12	13
<hr/>			
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	17	18	19
DTLS 1.3 (short header) (GCH)	12	13	14
<hr/>			
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 used 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-13 (work in progress), June 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-tls13-28 (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 Working Group
Internet-Draft
Intended status: Informational
Expires: May 6, 2021

J. Mattsson
F. Palombini
Ericsson AB
M. Vucinic
INRIA
November 02, 2020

Comparison of CoAP Security Protocols
draft-ietf-lwig-security-protocol-comparison-05

Abstract

This document analyzes and compares the sizes of key exchange flights and the 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, EDHOC, OSCORE, and Group OSCORE. The DTLS and TLS record layers are analyzed with and without 6LoWPAN-GHC compression. DTLS is analyzed with and without Connection ID.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Overhead of Key Exchange Protocols	3
2.1. Summary	4
2.2. DTLS 1.3	5
2.2.1. Message Sizes RPK + ECDHE	5
2.2.2. Message Sizes PSK + ECDHE	10
2.2.3. Message Sizes PSK	11
2.2.4. Cached Information	12
2.2.5. Resumption	13
2.2.6. Without Connection ID	14
2.2.7. DTLS Raw Public Keys	15
2.3. TLS 1.3	16
2.3.1. Message Sizes RPK + ECDHE	16
2.3.2. Message Sizes PSK + ECDHE	22
2.3.3. Message Sizes PSK	23
2.4. EDHOC	24
2.4.1. Message Sizes RPK	24
2.4.2. Summary	25
2.5. Conclusion	25
3. Overhead for Protection of Application Data	26
3.1. Summary	26
3.2. DTLS 1.2	28
3.2.1. DTLS 1.2	28
3.2.2. DTLS 1.2 with 6LoWPAN-GHC	28
3.2.3. DTLS 1.2 with Connection ID	29
3.2.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC	30
3.3. DTLS 1.3	30
3.3.1. DTLS 1.3	30
3.3.2. DTLS 1.3 with 6LoWPAN-GHC	31
3.3.3. DTLS 1.3 with Connection ID	31
3.3.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC	32
3.4. TLS 1.2	32
3.4.1. TLS 1.2	32
3.4.2. TLS 1.2 with 6LoWPAN-GHC	33
3.5. TLS 1.3	33
3.5.1. TLS 1.3	33
3.5.2. TLS 1.3 with 6LoWPAN-GHC	34
3.6. OSCORE	34
3.7. Group OSCORE	36
3.8. Conclusion	36
4. Security Considerations	37

5. IANA Considerations	37
6. Informative References	37
Acknowledgments	39
Authors' Addresses	39

1. Introduction

This document analyzes and compares the sizes of key exchange flights and the per-packet message size overheads when using different security protocols to secure CoAP over UDP [RFC7252] and TCP [RFC8323]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.ietf-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [RFC8446], EDHOC [I-D.ietf-lake-edhoc], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

The DTLS and TLS record layers are analyzed with and without 6LoWPAN-GHC compression. DTLS is analyzed with and without Connection ID [I-D.ietf-tls-dtls-connection-id]. Readers are expected to be familiar with some of the terms described in RFC 7925 [RFC7925], such as ICV. Section 2 compares the overhead of key exchange, while Section 3 covers the overhead for protection of application data.

2. Overhead of Key Exchange Protocols

This section analyzes and compares the sizes of key exchange flights for different protocols.

To enable a fair comparison between protocols, the following assumptions are made:

- o All the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (e.g. AES_128_CCM_8 or AES-CCM-16-64-128).
- o A minimum number of algorithms and cipher suites is offered. The algorithm used/offered are Curve25519, ECDSA with P-256, AES-CCM_8, SHA-256.
- o The length of key identifiers are 1 byte.
- o The length of connection identifiers are 1 byte.
- o DTLS RPK makes use of point compression, which saves 32 bytes.
- o DTLS handshake message fragmentation is not considered.
- o Only the DTLS mandatory extensions are considered, except for Connection ID.

Section 2.1 gives a short summary of the message overhead based on different parameters and some assumptions. The following sections detail the assumptions and the calculations.

2.1. Summary

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs of the same length, when Connection ID is used.

The EDHOC overhead is dependent on the key identifiers included. The following overheads apply for Sender IDs of the same length.

All the overhead are dependent on the tag length. The following overheads apply for tags of the same length.

Figure 1 compares the message sizes of EDHOC [I-D.ietf-lake-edhoc] with the DTLS 1.3 [I-D.ietf-tls-dtls13] and TLS 1.3 [RFC8446] handshakes with connection ID.

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE	150	373	213	736
DTLS 1.3 Cached X.509/RPK + ECDHE	182	347	213	742
DTLS 1.3 PSK + ECDHE	184	190	57	431
DTLS 1.3 PSK	134	150	57	341
EDHOC RPK + ECDHE	37	46	20	103
EDHOC X.509 + ECDHE	37	117	91	245

Figure 1: Comparison of message sizes in bytes with Connection ID

Figure 2 compares of message sizes of DTLS 1.3 [I-D.ietf-tls-dtls13] and TLS 1.3 [RFC8446] handshakes without connection ID.

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE	144	364	212	722
DTLS 1.3 PSK + ECDHE	178	183	56	417
DTLS 1.3 PSK	128	143	56	327
TLS 1.3 RPK + ECDHE	129	322	194	645
TLS 1.3 PSK + ECDHE	163	157	50	370
TLS 1.3 PSK	113	117	50	280

Figure 2: Comparison of message sizes in bytes without Connection ID

The details of the message size calculations are given in the following sections.

2.2. DTLS 1.3

This section gives an estimate of the message sizes of DTLS 1.3 with different authentication methods. Note that the examples in this section are not test vectors, the cryptographic parts are just replaced with byte strings of the same length, while other fixed length fields are replaced with arbitrary strings or omitted, in which case their length is indicated. Values that are not arbitrary are given in hexadecimal.

2.2.1. Message Sizes RPK + ECDHE

In this section, a Connection ID of 1 byte is used.

2.2.1.1. flight_1

Record Header - DTLSPlaintext (13 bytes):

16 fe fd EE EE SS SS SS SS SS SS LL LL

Handshake Header - Client Hello (10 bytes):

01 LL LL LL SS SS 00 00 00 LL LL LL

Legacy Version (2 bytes):

fe fd

Client Random (32 bytes):

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Legacy Session ID (1 bytes):

00

Legacy Cookie (1 bytes):
00

Cipher Suites (TLS_AES_128_CCM_8_SHA256) (4 bytes):
00 02 13 05

Compression Methods (null) (2 bytes):
01 00

Extensions Length (2 bytes):
LL LL

Extension - Supported Groups (x25519) (8 bytes):
00 0a 00 04 00 02 00 1d

Extension - Signature Algorithms (ecdsa_secp256r1_sha256)
(8 bytes):
00 0d 00 04 00 02 08 07

Extension - Key Share (42 bytes):
00 33 00 26 00 24 00 1d 00 20
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Extension - Supported Versions (1.3) (7 bytes):
00 2b 00 03 02 03 04

Extension - Client Certificate Type (Raw Public Key) (6 bytes):
00 13 00 01 01 02

Extension - Server Certificate Type (Raw Public Key) (6 bytes):
00 14 00 01 01 02

Extension - Connection Identifier (43) (6 bytes):
XX XX 00 02 01 42

13 + 10 + 2 + 32 + 1 + 1 + 4 + 2 + 2 + 8 + 8 + 42 + 7 + 6 + 6 + 6 = 150
bytes

DTLS 1.3 RPK + ECDHE flight_1 gives 150 bytes of overhead.

2.2.1.2. flight_2

Record Header - DTLSPlaintext (13 bytes):
16 fe fd EE EE SS SS SS SS SS SS LL LL

Handshake Header - Server Hello (10 bytes):
02 LL LL LL SS SS 00 00 00 LL LL LL

Legacy Version (2 bytes):
fe fd

Server Random (32 bytes):
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Legacy Session ID (1 bytes):
00

Cipher Suite (TLS_AES_128_CCM_8_SHA256) (2 bytes):
13 05

Compression Method (null) (1 bytes):
00

Extensions Length (2 bytes):
LL LL

Extension - Key Share (40 bytes):
00 33 00 24 00 1d 00 20
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Extension - Supported Versions (1.3) (6 bytes):
00 2b 00 02 03 04

Extension - Connection Identifier (43) (6 bytes):
XX XX 00 02 01 43

Record Header - DTLS Ciphertext, Full (6 bytes):
HH ES SS 43 LL LL

Handshake Header - Encrypted Extensions (10 bytes):
08 LL LL LL SS SS 00 00 00 LL LL LL

Extensions Length (2 bytes):
LL LL

Extension - Client Certificate Type (Raw Public Key) (6 bytes):
00 13 00 01 01 02

Extension - Server Certificate Type (Raw Public Key) (6 bytes):
00 14 00 01 01 02

Handshake Header - Certificate Request (10 bytes):
0d LL LL LL SS SS 00 00 00 LL LL LL

Request Context (1 bytes):
00

Extensions Length (2 bytes):
LL LL

Extension - Signature Algorithms (ecdsa_secp256r1_sha256)
(8 bytes):
00 0d 00 04 00 02 08 07

Handshake Header - Certificate (10 bytes):
0b LL LL LL SS SS 00 00 00 LL LL LL

Request Context (1 bytes):
00

Certificate List Length (3 bytes):
LL LL LL

Certificate Length (3 bytes):
LL LL LL

Certificate (59 bytes) // Point compression
....

Certificate Extensions (2 bytes):
00 00

Handshake Header - Certificate Verify (10 bytes):
0f LL LL LL SS SS 00 00 00 LL LL LL

Signature (68 bytes):
ZZ ZZ 00 40

Handshake Header - Finished (10 bytes):
14 LL LL LL SS SS 00 00 00 LL LL LL

Verify Data (32 bytes):
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Record Type (1 byte):
16

Auth Tag (8 bytes):
e0 8b 0e 45 5a 35 0a e5

13 + 102 + 6 + 24 + 21 + 78 + 78 + 42 + 1 + 8 = 373 bytes

DTLS 1.3 RPK + ECDHE flight_2 gives 373 bytes of overhead.

2.2.1.3. flight_3

Record Header (6 bytes) // DTLS Ciphertext, Full:
ZZ ES SS 42 LL LL

Handshake Header - Certificate (10 bytes):
0b LL LL LL SS SS XX XX XX LL LL LL

Request Context (1 bytes):
00

Certificate List Length (3 bytes):
LL LL LL

Certificate Length (3 bytes):
LL LL LL

Certificate (59 bytes) // Point compression
....

Certificate Extensions (2 bytes):
00 00

Handshake Header - Certificate Verify (10 bytes):
0f LL LL LL SS SS 00 00 00 LL LL LL

Signature (68 bytes):
04 03 LL LL //ecdsa_secp256r1_sha256
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 00 01 02 03 04 05 06 07 08 09 0a 0b
0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

Handshake Header - Finished (10 bytes):
14 LL LL LL SS SS 00 00 00 LL LL LL

Verify Data (32 bytes) // SHA-256:
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Record Type (1 byte):
16

Auth Tag (8 bytes) // AES-CCM_8:
00 01 02 03 04 05 06 07

6 + 78 + 78 + 42 + 1 + 8 = 213 bytes

DTLS 1.3 RPK + ECDHE flight_2 gives 213 bytes of overhead.

2.2.2. Message Sizes PSK + ECDHE

2.2.2.1. flight_1

The differences in overhead compared to Section 2.2.1.1 are:

The following is added:

+ Extension - PSK Key Exchange Modes (6 bytes):

00 2d 00 02 01 01

+ Extension - Pre Shared Key (48 bytes):

00 29 00 2F

00 0a 00 01 ID 00 00 00 00

00 21 20 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13

14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

The following is removed:

- Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

$150 + 6 + 48 - 8 - 6 - 6 = 184$ bytes

DTLS 1.3 PSK + ECDHE flight_1 gives 184 bytes of overhead.

2.2.2.2. flight_2

The differences in overhead compared to Section 2.2.1.2 are:

The following is added:

+ Extension - Pre Shared Key (6 bytes)

00 29 00 02 00 00

The following is removed:

- Handshake Message Certificate (78 bytes)
- Handshake Message CertificateVerify (78 bytes)
- Handshake Message CertificateRequest (21 bytes)
- Extension - Client Certificate Type (Raw Public Key) (6 bytes)
- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

$373 - 78 - 78 - 21 - 6 - 6 + 6 = 190$ bytes

DTLS 1.3 PSK + ECDHE flight_2 gives 190 bytes of overhead.

2.2.2.3. flight_3

The differences in overhead compared to Section 2.2.1.3 are:

The following is removed:

- Handshake Message Certificate (78 bytes)
- Handshake Message Certificate Verify (78 bytes)

In total:

$213 - 78 - 78 = 57$ bytes

DTLS 1.3 PSK + ECDHE flight_3 gives 57 bytes of overhead.

2.2.3. Message Sizes PSK

2.2.3.1. flight_1

The differences in overhead compared to Section 2.2.2.1 are:

The following is removed:

- Extension - Supported Groups (x25519) (8 bytes)
- Extension - Key Share (42 bytes)

In total:

$184 - 8 - 42 = 134$ bytes

DTLS 1.3 PSK flight_1 gives 134 bytes of overhead.

2.2.3.2. flight_2

The differences in overhead compared to Section 2.2.2.2 are:

The following is removed:

- Extension - Key Share (40 bytes)

In total:

$190 - 40 = 150$ bytes

DTLS 1.3 PSK flight_2 gives 150 bytes of overhead.

2.2.3.3. flight_3

There are no differences in overhead compared to Section 2.2.2.3.

DTLS 1.3 PSK flight_3 gives 57 bytes of overhead.

2.2.4. Cached Information

In this section, we consider the effect of [RFC7924] on the message size overhead.

Cached information together with server X.509 can be used to move bytes from flight #2 to flight #1 (cached RPK increases the number of bytes compared to cached X.509).

The differences compared to Section 2.2.1 are the following.

For the flight #1, the following is added:

- + Extension - Client Cached Information (39 bytes):

```
00 19 LL LL LL LL
01 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f
```

And the following is removed:

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

Giving a total of:

$150 + 33 = 183$ bytes

For the flight #2, the following is added:

+ Extension - Server Cashed Information (7 bytes):
00 19 LL LL LL LL 01

And the following is removed:

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)
- Server Certificate (59 bytes -> 32 bytes)

Giving a total of:

$373 - 26 = 347$ bytes

A summary of the calculation is given in Figure 3.

Flight	#1	#2	#3	Total
DTLS 1.3 Cached X.509/RPK + ECDHE	183	347	213	743
DTLS 1.3 RPK + ECDHE	150	373	213	736

Figure 3: Comparison of message sizes in bytes for DTLS 1.3 RPK + ECDH with and without cached X.509

2.2.5. Resumption

To enable resumption, a 4th flight (New Session Ticket) is added to the PSK handshake.

Record Header - DTLS Ciphertext, Full (6 bytes):
 HH ES SS 43 LL LL

Handshake Header - New Session Ticket (10 bytes):
 04 LL LL LL SS SS 00 00 00 LL LL LL

Ticket Lifetime (4 bytes):
 00 01 02 03

Ticket Age Add (4 bytes):
 00 01 02 03

Ticket Nonce (2 bytes):
 01 00

Ticket (6 bytes):
 00 04 ID ID ID ID

Extensions (2 bytes):
 00 00

Auth Tag (8 bytes) // AES-CCM_8:
 00 01 02 03 04 05 06 07

$6 + 10 + 4 + 4 + 2 + 6 + 2 + 8 = 42$ bytes

The initial handshake when resumption is enabled is just a PSK handshake with $134 + 150 + 57 + 42 = 383$ bytes.

2.2.6. Without Connection ID

Without a Connection ID the DTLS 1.3 flight sizes changes as follows.

DTLS 1.3 Flight #1: -6 bytes
 DTLS 1.3 Flight #2: -7 bytes
 DTLS 1.3 Flight #3: -1 byte

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE (no cid)	144	364	212	722
DTLS 1.3 PSK + ECDHE (no cid)	178	183	56	417
DTLS 1.3 PSK (no cid)	128	143	56	327

Figure 4: Comparison of message sizes in bytes for DTLS 1.3 without Connection ID

2.2.7. DTLS Raw Public Keys

TODO

2.2.7.1. SubjectPublicKeyInfo without point compression

0x30 // Sequence

0x59 // Size 89

0x30 // Sequence

0x13 // Size 19

0x06 0x07 0x2A 0x86 0x48 0xCE 0x3D 0x02 0x01

// OID 1.2.840.10045.2.1 (ecPublicKey)

0x06 0x08 0x2A 0x86 0x48 0xCE 0x3D 0x03 0x01 0x07

// OID 1.2.840.10045.3.1.7 (secp256r1)

0x03 // Bit string

0x42 // Size 66

0x00 // Unused bits 0

0x04 // Uncompressed

..... 64 bytes X and Y

Total of 91 bytes

2.2.7.2. SubjectPublicKeyInfo with point compression

0x30 // Sequence

0x59 // Size 89

0x30 // Sequence

0x13 // Size 19

0x06 0x07 0x2A 0x86 0x48 0xCE 0x3D 0x02 0x01

// OID 1.2.840.10045.2.1 (ecPublicKey)

0x06 0x08 0x2A 0x86 0x48 0xCE 0x3D 0x03 0x01 0x07

// OID 1.2.840.10045.3.1.7 (secp256r1)

0x03 // Bit string

0x42 // Size 66

0x00 // Unused bits 0

0x03 // Compressed

..... 32 bytes X

Total of 59 bytes

2.3. TLS 1.3

In this section, the message sizes are calculated for TLS 1.3. The major changes compared to DTLS 1.3 are that the record header is smaller, the handshake headers is smaller, and that Connection ID is not supported. Recently, additional work has taken shape with the goal to further reduce overhead for TLS 1.3 (see [I-D.rescorla-tls-ctls]).

TLS Assumptions:

- o Minimum number of algorithms and cipher suites offered
- o Curve25519, ECDSA with P-256, AES-CCM_8, SHA-256
- o Length of key identifiers: 1 bytes
- o TLS RPK with point compression (saves 32 bytes)
- o Only mandatory TLS extensions

For the PSK calculations, [Ulfheim-TLS13] was a useful resource, while for RPK calculations we followed the work of [IoT-Cert].

2.3.1. Message Sizes RPK + ECDHE

2.3.1.1. flight_1

Record Header - TLSPlaintext (5 bytes):

16 03 03 LL LL

Handshake Header - Client Hello (4 bytes):

01 LL LL LL

Legacy Version (2 bytes):

03 03

Client Random (32 bytes):

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Legacy Session ID (1 bytes):

00

Cipher Suites (TLS_AES_128_CCM_8_SHA256) (4 bytes):

00 02 13 05

Compression Methods (null) (2 bytes):

01 00

Extensions Length (2 bytes):

LL LL

Extension - Supported Groups (x25519) (8 bytes):

00 0a 00 04 00 02 00 1d

Extension - Signature Algorithms(ecdsa_secp256r1_sha256) (8 bytes):

00 0d 00 04 00 02 08 07

Extension - Key Share (42 bytes):

00 33 00 26 00 24 00 1d 00 20
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Extension - Supported Versions (1.3) (7 bytes):

00 2b 00 03 02 03 04

Extension - Client Certificate Type (Raw Public Key) (6 bytes):

00 13 00 01 01 02

Extension - Server Certificate Type (Raw Public Key) (6 bytes):

00 14 00 01 01 02

5 + 4 + 2 + 32 + 1 + 4 + 2 + 2 + 8 + 8 + 42 + 7 + 6 + 6 = 129 bytes

TLS 1.3 RPK + ECDHE flight_1 gives 129 bytes of overhead.

2.3.1.2. flight_2

Record Header - TLSPlaintext (5 bytes):
16 03 03 LL LL

Handshake Header - Server Hello (4 bytes):
02 LL LL LL

Legacy Version (2 bytes):
fe fd

Server Random (32 bytes):
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Legacy Session ID (1 bytes):
00

Cipher Suite (TLS_AES_128_CCM_8_SHA256) (2 bytes):
13 05

Compression Method (null) (1 bytes):
00

Extensions Length (2 bytes):
LL LL

Extension - Key Share (40 bytes):
00 33 00 24 00 1d 00 20
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Extension - Supported Versions (1.3) (6 bytes):
00 2b 00 02 03 04

Record Header - TLSCiphertext (5 bytes):
17 03 03 LL LL

Handshake Header - Encrypted Extensions (4 bytes):
08 LL LL LL

Extensions Length (2 bytes):
LL LL

Extension - Client Certificate Type (Raw Public Key) (6 bytes):
00 13 00 01 01 02

Extension - Server Certificate Type (Raw Public Key) (6 bytes):

00 14 00 01 01 02

Handshake Header - Certificate Request (4 bytes):

0d LL LL LL

Request Context (1 bytes):

00

Extensions Length (2 bytes):

LL LL

Extension - Signature Algorithms(ecdsa_secp256r1_sha256) (8 bytes):

00 0d 00 04 00 02 08 07

Handshake Header - Certificate (4 bytes):

0b LL LL LL

Request Context (1 bytes):

00

Certificate List Length (3 bytes):

LL LL LL

Certificate Length (3 bytes):

LL LL LL

Certificate (59 bytes) // Point compression

....

Certificate Extensions (2 bytes):

00 00

Handshake Header - Certificate Verify (4 bytes):

0f LL LL LL

Signature (68 bytes):

ZZ ZZ 00 40

Handshake Header - Finished (4 bytes):

14 LL LL LL

Verify Data (32 bytes):

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Record Type (1 byte):

16

Auth Tag (8 bytes):

e0 8b 0e 45 5a 35 0a e5

$5 + 90 + 5 + 18 + 15 + 72 + 72 + 36 + 1 + 8 = 322$ bytes

TLS 1.3 RPK + ECDHE flight_2 gives 322 bytes of overhead.

2.3.1.3. flight_3

Record Header - TLSCiphertext (5 bytes):

17 03 03 LL LL

Handshake Header - Certificate (4 bytes):

0b LL LL LL

Request Context (1 bytes):

00

Certificate List Length (3 bytes):

LL LL LL

Certificate Length (3 bytes):

LL LL LL

Certificate (59 bytes) // Point compression

....

Certificate Extensions (2 bytes):

00 00

Handshake Header - Certificate Verify (4 bytes):

0f LL LL LL

Signature (68 bytes):

04 03 LL LL //ecdsa_secp256r1_sha256

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 00 01 02 03 04 05 06 07 08 09 0a 0b
0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

Handshake Header - Finished (4 bytes):

14 LL LL LL

Verify Data (32 bytes) // SHA-256:

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f

Record Type (1 byte)

16

Auth Tag (8 bytes) // AES-CCM_8:

00 01 02 03 04 05 06 07

5 + 72 + 72 + 36 + 1 + 8 = 194 bytes

TLS 1.3 RPK + ECDHE flight_3 gives 194 bytes of overhead.

2.3.2. Message Sizes PSK + ECDHE

2.3.2.1. flight_1

The differences in overhead compared to Section 2.3.1.3 are:

The following is added:

+ Extension - PSK Key Exchange Modes (6 bytes):

00 2d 00 02 01 01

+ Extension - Pre Shared Key (48 bytes):

00 29 00 2F

00 0a 00 01 ID 00 00 00 00

00 21 20 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13

14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

The following is removed:

- Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

$129 + 6 + 48 - 8 - 6 - 6 = 163$ bytes

TLS 1.3 PSK + ECDHE flight_1 gives 166 bytes of overhead.

2.3.2.2. flight_2

The differences in overhead compared to Section 2.3.1.2 are:

The following is added:

+ Extension - Pre Shared Key (6 bytes)

00 29 00 02 00 00

The following is removed:

- Handshake Message Certificate (72 bytes)
- Handshake Message CertificateVerify (72 bytes)
- Handshake Message CertificateRequest (15 bytes)
- Extension - Client Certificate Type (Raw Public Key) (6 bytes)
- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

$322 - 72 - 72 - 15 - 6 - 6 + 6 = 157$ bytes

TLS 1.3 PSK + ECDHE flight_2 gives 157 bytes of overhead.

2.3.2.3. flight_3

The differences in overhead compared to Section 2.3.1.3 are:

The following is removed:

- Handshake Message Certificate (72 bytes)
- Handshake Message Certificate Verify (72 bytes)

In total:

$194 - 72 - 72 = 50$ bytes

TLS 1.3 PSK + ECDHE flight_3 gives 50 bytes of overhead.

2.3.3. Message Sizes PSK

2.3.3.1. flight_1

The differences in overhead compared to Section 2.3.2.1 are:

The following is removed:

- Extension - Supported Groups (x25519) (8 bytes)
- Extension - Key Share (42 bytes)

In total:

$163 - 8 - 42 = 113$ bytes

TLS 1.3 PSK flight_1 gives 116 bytes of overhead.

2.3.3.2. flight_2

The differences in overhead compared to Section 2.3.2.2 are:

The following is removed:

- Extension - Key Share (40 bytes)

In total:

157 - 40 = 117 bytes

TLS 1.3 PSK flight_2 gives 117 bytes of overhead.

2.3.3.3. flight_3

There are no differences in overhead compared to Section 2.3.2.3.

TLS 1.3 PSK flight_3 gives 57 bytes of overhead.

2.4. EDHOC

This section gives an estimate of the message sizes of EDHOC with authenticated with static Diffie-Hellman keys. All examples are given in CBOR diagnostic notation and hexadecimal, and are based on the test vectors in Appendix B.2 of [I-D.ietf-lake-edhoc].

2.4.1. Message Sizes RPK

2.4.1.1. message_1

```
message_1 = (  
  13,  
  0,  
  h'8D3EF56D1B750A4351D68AC250A0E883790EFC80A538A444EE9E2B57E244  
    1A7C',  
  -2  
)
```

```
message_1 (37 bytes):  
0d 00 58 20 8d 3e f5 6d 1b 75 0a 43 51 d6 8a c2 50 a0 e8 83  
79 0e fc 80 a5 38 a4 44 ee 9e 2b 57 e2 44 1a 7c 21
```

2.4.1.2. message_2

```

message_2 = (
  h'52FBA0BDC8D953DD86CE1AB2FD7C05A4658C7C30AFDBFC3301047069451B
  AF35',
  8,
  h'DCF6FE9C524C22454DEB'
)

```

```

message_2 (46 bytes):
58 20 52 fb a0 bd c8 d9 53 dd 86 ce 1a b2 fd 7c 05 a4 65 8c
7c 30 af db fc 33 01 04 70 69 45 1b af 35 08 4a dc f6 fe 9c
52 4c 22 45 4d eb

```

2.4.1.3. message_3

```

message_3 = (
  8,
  h'53C3991999A5FFB86921E99B607C067770E0'
)

```

```

message_3 (20 bytes):
08 52 53 c3 99 19 99 a5 ff b8 69 21 e9 9b 60 7c 06 77 70 e0

```

2.4.2. Summary

The typical message sizes for the previous example and for an example of EDHOC authenticated with signature keys and X.509 certificates based on Appendix B.1 of [I-D.ietf-lake-edhoc] are summarized in Figure 5.

	RPK	x5t
message_1	37	37
message_2	46	117
message_3	20	91
Total	103	245

Figure 5: Typical message sizes in bytes

2.5. Conclusion

To do a fair comparison, one has to choose a specific deployment and look at the topology, the whole protocol stack, frame sizes (e.g. 51 or 128 bytes), how and where in the protocol stack fragmentation is

done, and the expected packet loss. Note that the number of bytes in each frame that is available for the key exchange protocol may depend on the underlying protocol layers as well as on the number of hops in multi-hop networks. The packet loss may depend on how many other devices are transmitting at the same time, and may increase during network formation. The total overhead will be larger due to mechanisms for fragmentation, retransmission, and packet ordering. The overhead of fragmentation is roughly proportional to the number of fragments, while the expected overhead due to retransmission in noisy environments is a superlinear function of the flight sizes.

3. Overhead for Protection of Application Data

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (e.g. AES_128_CCM_8 or AES-CCM-16-64), 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 DTLS 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 calculations are only examples.

Section 3.1 gives a short summary of the message overhead based on different parameters and some assumptions. The following sections detail the assumptions and the calculations.

3.1. Summary

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	11	12	12
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	12	13	13
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 6: 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	11	12	13
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	12	13	14
OSCORE request	13	14	15
OSCORE response	11	11	11

Figure 7: 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	3	4
TLS 1.2	13	9
TLS 1.3	6	7
OSCORE request	5	
OSCORE response	3	

Figure 8: Overhead (excluding ICV) in bytes
(Connection/Sender ID = '', Sequence Number = '05')

3.2. DTLS 1.2

3.2.1. 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.

3.2.2. DTLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with 6LoWPAN-GHC [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 used 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.

3.2.3. 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 record layer 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.

3.2.4. 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 6LoWPAN-GHC [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 used 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.

3.3. DTLS 1.3

3.3.1. 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 into the first byte containing signalling bits, optional omission of length, reduction of sequence number into a 1 or 2-bytes field.

Only the minimal header format for DTLS 1.3 is analyzed (see Figure 4 of [I-D.ietf-tls-dtls13]). The minimal header format omit the length field and only a 1-byte field is used to carry the 8 low order bits of the sequence number

DTLS 1.3 record layer (17 bytes, 11 bytes overhead):
21 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

First byte (including epoch):

21

Sequence number:

05

Ciphertext (including encrypted content type):

ae a0 15 56 67 92 ec

ICV:

4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 11 bytes overhead.

3.3.2. DTLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

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

Compressed DTLS 1.3 record layer (18 bytes, 12 bytes overhead):

11 21 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb
35 b9

Compressed DTLS 1.3 record layer header and nonce:

11 21 05

Ciphertext (including encrypted content type):

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, no length) gives 12 bytes overhead.

3.3.3. 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].

In this example, the length field is omitted, and the 1-byte field is used for the sequence number. The minimal DTLSCiphertext structure is used (see Figure 4 of [I-D.ietf-tls-dtls13]), with the addition of the Connection ID field.

DTLS 1.3 record layer (18 bytes, 12 bytes overhead):
31 42 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

First byte (including epoch):

31

Connection ID:

42

Sequence number:

05

Ciphertext (including encrypted content type):

ae a0 15 56 67 92 ec

ICV:

4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 with Connection ID gives 12 bytes overhead.

3.3.4. 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 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

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

Compressed DTLS 1.3 record layer (19 bytes, 13 bytes overhead):

12 31 05 42 ae a0 15 56 67 92 ec 4d ff 8a 24 e4
cb 35 b9

Compressed DTLS 1.3 record layer header and nonce:

12 31 05 42

Ciphertext (including encrypted content type):

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, no length) gives 13 bytes overhead.

3.4. TLS 1.2

3.4.1. 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.

3.4.2. TLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.2 [RFC5246] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used 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.

3.5. TLS 1.3

3.5.1. TLS 1.3

This section analyzes the overhead of TLS 1.3 [RFC8446]. 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 content type):

ae a0 15 56 67 92 ec

ICV:

4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 14 bytes overhead.

3.5.2. TLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.3 [RFC8446] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used 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 content type):

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.

3.6. OSCORE

This section analyzes the overhead of OSCORE [RFC8613].

The below calculation Option Delta = '9', Sender ID = '' (empty string), and Sequence Number = '05', and is only an example. Note that Sender ID = '' (empty string) can only be used by one client per server.

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.

3.7. Group OSCORE

This section analyzes the overhead of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

TODO

3.8. Conclusion

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 has quite a small overhead. OSCORE and DTLS 1.3 (using the minimal structure) format have 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 used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

New security protocols like OSCORE, TLS 1.3, and DTLS 1.3 have much lower overhead than DTLS 1.2 and TLS 1.2. The overhead is even smaller than DTLS 1.2 and TLS 1.2 over 6LoWPAN with compression, and therefore the small overhead is achieved even on deployments without 6LoWPAN or 6LoWPAN without compression. OSCORE is lightweight because it makes use of CoAP, CBOR, and COSE, which were designed to have as low overhead as possible.

Note that the compared protocols have slightly different use cases. TLS and DTLS are designed for the transport layer and are terminated in CoAP proxies. OSCORE is designed for the application layer and protects information end-to-end between the CoAP client and the CoAP server. Group OSCORE is designed for group communication and protects information between a CoAP client and any number of CoAP servers.

4. Security Considerations

This document is purely informational.

5. IANA Considerations

This document has no actions for IANA.

6. Informative References

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP",
draft-ietf-core-oscore-groupcomm-09 (work in progress),
June 2020.

[I-D.ietf-lake-edhoc]

Selander, G., Mattsson, J., and F. Palombini, "Ephemeral
Diffie-Hellman Over COSE (EDHOC)", draft-ietf-lake-
edhoc-01 (work in progress), August 2020.

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

Rescorla, E., Tschofenig, H., and T. Fossati, "Connection
Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-
id-07 (work in progress), October 2019.

[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-dtls13-38 (work in progress), May
2020.

- [I-D.rescorla-tls-ctls]
Rescorla, E., Barnes, R., and H. Tschofenig, "Compact TLS 1.3", draft-rescorla-tls-ctls-04 (work in progress), March 2020.
- [IoT-Cert]
Forsby, F., "Digital Certificates for the Internet of Things", June 2017, <<https://kth.diva-portal.org/smash/get/diva2:1153958/FULLTEXT01.pdf>>.
- [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>>.
- [RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <<https://www.rfc-editor.org/info/rfc7924>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [Ulfheim-TLS13]
Driscoll, M., "Every Byte Explained The Illustrated TLS 1.3 Connection", March 2018, <<https://tls13.ulfheim.net>>.

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 Preuss Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Malisa Vucinic
INRIA

Email: malisa.vucinic@inria.fr

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 (CNs), 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.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Characteristics of CNNs relevant for TCP	4
2.1. Network and link properties	4
2.2. Usage scenarios	5
2.3. Communication and traffic patterns	6
3. TCP implementation and configuration in CNNs	6
3.1. Addressing path properties	7
3.1.1. Maximum Segment Size (MSS)	7
3.1.2. Explicit Congestion Notification (ECN)	8
3.1.3. Explicit loss notifications	9
3.2. TCP guidance for single-MSS stacks	9
3.2.1. Single-MSS stacks - benefits and issues	9
3.2.2. TCP options for single-MSS stacks	10
3.2.3. Delayed Acknowledgments for single-MSS stacks	10
3.2.4. RTO calculation for single-MSS stacks	11
3.3. General recommendations for TCP in CNNs	12
3.3.1. Loss recovery and congestion/flow control	12
3.3.1.1. Selective Acknowledgments (SACK)	13
3.3.2. Delayed Acknowledgments	13
3.3.3. Initial Window	14
4. TCP usage recommendations in CNNs	14
4.1. TCP connection initiation	14
4.2. Number of concurrent connections	15
4.3. TCP connection lifetime	15
5. Security Considerations	17
6. Acknowledgments	18
7. Annex. TCP implementations for constrained devices	18
7.1. uIP	19
7.2. lwIP	19
7.3. RIOT	19
7.4. TinyOS	20
7.5. FreeRTOS	20
7.6. uC/OS	20
7.7. Summary	21
8. Annex. Changes compared to previous versions	22
8.1. Changes between -00 and -01	22
8.2. Changes between -01 and -02	22
8.3. Changes between -02 and -03	22
8.4. Changes between -03 and -04	23

8.5.	Changes between -04 and -05	23
8.6.	Changes between -05 and -06	23
8.7.	Changes between -06 and -07	23
8.8.	Changes between -07 and -08	23
8.9.	Changes between -08 and -09	23
8.10.	Changes between -09 and -10	24
8.11.	Changes between -10 and -11	24
8.12.	Changes between -11 and -12	24
8.13.	Changes between -12 and -13	24
9.	References	24
9.1.	Normative References	24
9.2.	Informative References	25
	Authors' Addresses	30

1. Introduction

The Internet Protocol suite is being used for connecting Constrained-Node Networks (CNs) to the Internet, enabling the so-called Internet of Things (IoT) [RFC7228]. In order to meet the requirements that stem from CNs, 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 CNs 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.

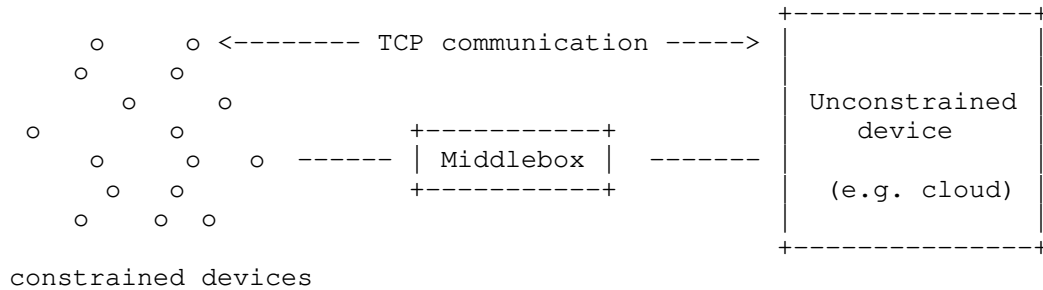


Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

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:

- o 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.
- o Request-response patterns: An IoT device receiving a request from the other endpoint, which triggers a response from the IoT device.
- o 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 assess 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 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, Gorrry 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 outgoing 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/retr	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

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

8.2. Changes between -01 and -02

- o Added text to the Introduction section, and a reference, on traditional bad perception of TCP for IoT
- o Added sections on FreeRTOS and uC/OS
- o Updated TinyOS section
- o Updated summary table
- o 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

- o Rewording to better explain the benefit of ECN
- o Additional context information on the surveyed implementations
- o Added details, but removed "Data size" row, in the summary table

- o Added discussion on shrew attacks
- 8.4. Changes between -03 and -04
- o Addressing the remaining TODOs
 - o Alignment of the wording on TCP "keep-alives" with related discussions in the IETF transport area
 - o Added further discussion on delayed ACKs
 - o Removed OpenWSN section from the Annex
- 8.5. Changes between -04 and -05
- o Addressing comments by Yoshifumi Nishida
 - o Removed mentioning MD5 as an example (comment by David Black)
 - o Added memory footprint details of TCP implementations (Contiki-NG and lwIP 2.1.2) provided by Rahul Jadhav in the Annex
 - o Addressed comments by Ilpo Jarvinen throughout the whole document
 - o Improved the RIOT section in the Annex, based on feedback from Emmanuel Baccelli
- 8.6. Changes between -05 and -06
- o Incorporated suggestions by Stuart Cheshire
- 8.7. Changes between -06 and -07
- o Addressed comments by Gorrry Fairhurst
- 8.8. Changes between -07 and -08
- o 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
- o Addressed second round of comments by Ilpo Jarvinen and Markku Kojo, based on the previous draft update.

8.10. Changes between -09 and -10

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

8.11. Changes between -10 and -11

- o Addressed a comment by Ted Lemon on MSS advice.

8.12. Changes between -11 and -12

- o Addressed comments from IESG and various directorates.

8.13. Changes between -12 and -13

- o Fixed two typos.
- o 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., Dukkkipati, 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 RIOT Approach to Ubiquitous Networking for the IoT", 2018.
- [HomeGateway] Haetoeinen, 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 System for 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

Light-Weight Implementation Guidance (lwig)
Internet-Draft
Intended status: Informational
Expires: December 23, 2018

D. Migault
Ericsson
T. Guggemos
LMU Munich
June 21, 2018

Minimal ESP
draft-mglt-lwig-minimal-esp-06

Abstract

This document describes a minimal implementation of the IP Encapsulation Security Payload (ESP) defined in RFC 4303. Its purpose is to enable implementation of ESP with a minimal set of options to remain compatible with ESP as described in RFC 4303. A minimal version of ESP is not intended to become a replacement of the RFC 4303 ESP, but instead to enable a limited implementation to interoperate with implementations of RFC 4303 ESP.

This document describes what is required from RFC 4303 ESP as well as various ways to optimize compliance with RFC 4303 ESP.

This document does not update or modify RFC 4303, but provides a compact description of how to implement the minimal version of the protocol. If this document and RFC 4303 conflicts then RFC 4303 is the authoritative description.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 23, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

1. Requirements notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Introduction

ESP [RFC4303] is part of the IPsec suite protocol [RFC4301]. IPsec is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity) and limited traffic flow confidentiality.

Figure 1 describes an ESP Packet. Currently ESP is implemented in the kernel of major multi purpose Operating Systems (OS). The ESP and IPsec suite is usually implemented in a complete way to fit multiple purpose usage of these OS. However, completeness of the IPsec suite as well as multi purpose scope of these OS is often performed at the expense of resources, or a lack of performance. As a result, constrained devices are likely to have their own implementation of ESP optimized and adapted to their specificities. With the adoption of IPsec by IoT devices with minimal IKEv2 [RFC7815] and ESP Header Compression (EHC) with [I-D.mglt-ipsecme-diet-esp] or [I-D.mglt-ipsecme-ikev2-diet-esp-extension], it becomes crucial that ESP implementation designed for constraint devices remain interoperable with the standard ESP implementation to avoid a fragmented usage of ESP. This document describes the the minimal properties and ESP implementation needs to meet.

For each field of the ESP packet represented in Figure 1 this document provides recommendations and guidance for minimal implementations. The primary purpose of Minimal ESP is to remain interoperable with other nodes implementing RFC 4303 ESP, while limiting the standard complexity of the implementation.

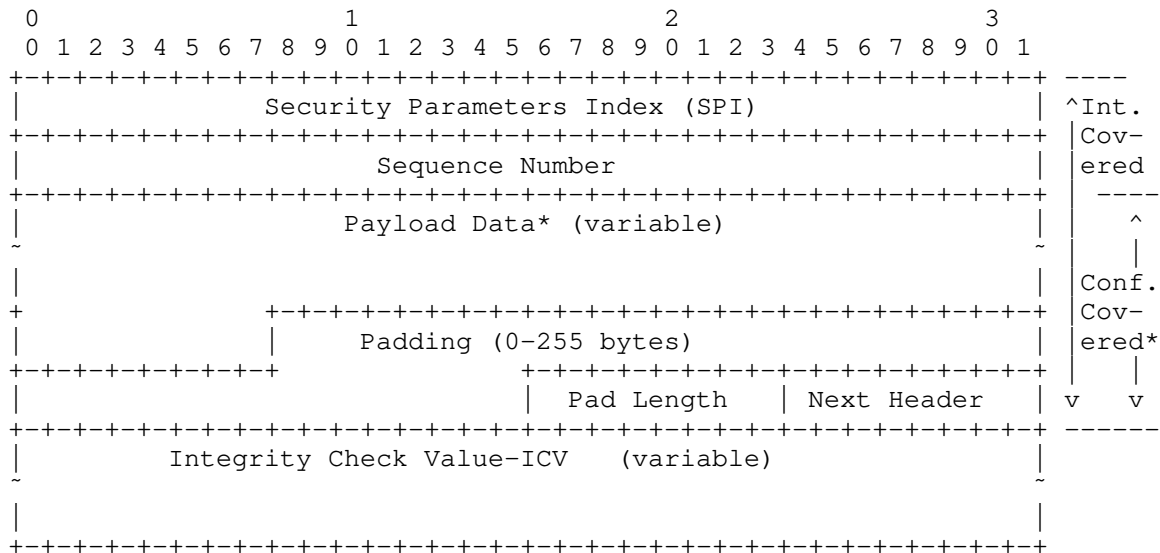


Figure 1: ESP Packet Description

3. Security Parameter Index (SPI) (32 bit)

According to the [RFC4303], the SPI is a mandatory 32 bits field and is not allowed to be removed.

The SPI has a local significance to index the Security Association (SA). From [RFC4301] section 4.1, nodes supporting only unicast communications can index their SA only using the SPI. On the other hand, nodes supporting multicast communications must also use the IP addresses and thus SA lookup needs to be performed using the longest match.

For nodes supporting only unicast communications, it is RECOMMENDED to index SA with the SPI only. Some other local constraints on the node may require a combination of the SPI as well as other parameters to index the SA.

It is RECOMMENDED to randomly generate the SPI indexing each inbound session. A random generation provides a stateless way to generate the SPIs, while keeping the probability of collision between SPIs

relatively low. In case of collision, the SPI is simply re-generated.

However, for some constrained nodes, generating a random SPI may consume too much resource, in which case SPI can be generated using predictable functions or even a fix value. In fact, the SPI does not need to be random.

When a constrained node uses fix value for SPIs, it imposes some limitations on the number of inbound SA. This limitation can be alleviated by how the SA look up is performed. When fix SPI are used, it is RECOMMENDED that the constrained node has as many SPI values as ESP session per host IP address, and that SA lookup includes the IP addresses.

Note that SPI value is used only for inbound traffic, as such the SPI negotiated with IKEv2 [RFC7296] or [RFC7815] by a peer, is the value used by the remote peer when it sends traffic. As SPI are only used for inbound traffic by the peer, this allows each peer to manage the set of SPIs used for its inbound traffic.

The use of fix SPI should not be considered as a way to avoid strong random generators. Such generator will be required in order to provide strong cryptographic protection and follow the randomness requirements for security described in [RFC4086]. Instead, the use of a fix SPI should only be considered as a way to overcome the resource limitations of the node, when this is feasible.

The use of a limited number of fix SPI also comes with security or privacy drawbacks. Typically, a passive attacker may derive information such as the number of constrained devices connecting the remote peer, and in conjunction with data rate, the attacker may eventually determine the application the constrained device is associated to. In addition, if the fix value SPI is fixed by a manufacturer or by some software application, the SPI may leak in an obvious way the type of sensor, the application involved or the model of the constrained device. As a result, the use of a unpredictable SPI is preferred to provide better privacy.

As far as security is concerned, revealing the type of application or model of the constrained device could be used to identify the vulnerabilities the constrained device is subject to. This is especially sensitive for constrained device where patches or software updates will be challenging to operate. As a result, these devices may remain vulnerable for a relatively long period. In addition, predictable SPI enable an attacker to forge packets with a valid SPI. Such packet will not be rejected due to an SPI mismatch, but instead

after the signature check which requires more resource and thus make DoS more efficient, especially for devices powered by batteries.

Values 0-255 SHOULD NOT be used. Values 1-255 are reserved and 0 is only allowed to be used internal and it MUST NOT be send on the wire.

[RFC4303] mentions :

"The SPI is an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound. The SPI field is mandatory. [...]"

"For a unicast SA, the SPI can be used by itself to specify an SA, or it may be used in conjunction with the IPsec protocol type (in this case ESP). Because the SPI value is generated by the receiver for a unicast SA, whether the value is sufficient to identify an SA by itself or whether it must be used in conjunction with the IPsec protocol value is a local matter. This mechanism for mapping inbound traffic to unicast SAs MUST be supported by all ESP implementations."

4. Sequence Number(SN) (32 bit)

According to [RFC4303], the sequence number is a mandatory 32 bits field in the packet.

The SN is set by the sender so the receiver can implement anti-replay protection. The SN is derived from any strictly increasing function that guarantees: if packet B is sent after packet A, then SN of packet B is strictly greater then the SN of packet A.

Some constrained devices may establish communication with specific devices, like a specific gateway, or nodes similar to them. As a result, the sender may know whereas the receiver implements anti-replay protection or not. Even though the sender may know the receiver does not implement anti replay protection, the sender MUST implement a always increasing function to generate the SN.

Usually, SN is generated by incrementing a counter for each packet sent. A constrained device may avoid maintaining this context. If the device has a clock, it may use the time indicated by the clock has a SN. This guarantees a strictly increasing function, and avoid storing any additional values or context related to the SN. When the use of a clock is considered, one should take care that packets associated to a given SA are not sent with the same time value.

[RFC4303] mentions :

"This unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number. For a unicast SA or a single-sender multicast SA, the sender MUST increment this field for every transmitted packet. Sharing an SA among multiple senders is permitted, though generally not recommended. [...] The field is mandatory and MUST always be present even if the receiver does not elect to enable the anti-replay service for a specific SA."

5. Padding

The purpose of padding is to respect the 32 bit alignment of ESP. ESP MUST have at least one padding byte Pad Length that indicates the padding length. ESP padding bytes are generated by a succession of unsigned bytes starting with 1, 2, 3 with the last byte set to Pad Length, where Pad Length designates the length of the padding bytes.

Checking the padding structure is not mandatory, so the constrained device may not proceed to such checks, however, in order to interoperate with existing ESP implementations, it MUST build the padding bytes as recommended by ESP.

In some situation the padding bytes may take a fix value. This would typically be the case when the Data Payload is of fix size.

[RFC4303] mentions :

"If Padding bytes are needed but the encryption algorithm does not specify the padding contents, then the following default processing MUST be used. The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, When this padding scheme is employed, the receiver SHOULD inspect the Padding field. (This scheme was selected because of its relative simplicity, ease of implementation in hardware, and because it offers limited protection against certain forms of "cut and paste" attacks in the absence of other integrity measures, if the receiver checks the padding values upon decryption.)"

ESP [RFC4303] also provides Traffic Flow Confidentiality (TFC) as a way to perform padding to hide traffic characteristics, which differs from respecting a 32 bit alignment. TFC is not mandatory and MUST be negotiated with the SA management protocol. As a result, TFC is not expected to be supported by a minimal ESP implementation. On the other hand, disabling TFC should be carefully measured and understood as it exposes the node to traffic shaping. This could expose the application as well as the devices used to a passive monitoring

attacker. Such information could be used by the attacker in case a vulnerability is disclosed on the specific device. In addition, some application use - such as health applications - may also reveal important privacy oriented informations.

Some constrained nodes that have limited battery life time may also prefer avoiding sending extra padding bytes. However the same nodes may also be very specific to an application and device. As a result, they are also likely to be the main target for traffic shaping. In most cases, the payload carried by these nodes is quite small, and the standard padding mechanism may also be used as an alternative to TFC, with a sufficient trade off between the require energy to send additional payload and the exposure to traffic shaping attacks.

6. Next Header (8 bit)

According to [RFC4303], the Next Header is a mandatory 8 bits field in the packet. Next header is intended to specify the data contained in the payload as well as dummy packet. In addition, the Next Header may also carry an indication on how to process the packet [I-D.nikander-esp-beet-mode].

The ability to generate and receive dummy packet is required by [RFC4303]. For interoperability, it is RECOMMENDED a minimal ESP implementation discards dummy packets. Note that such recommendation only applies for nodes receiving packets, and that nodes designed to only send data may not implement this capability.

As the generation of dummy packets is subject to local management and based on a per-SA basis, a minimal ESP implementation may not generate such dummy packet. More especially, in constrained environments sending dummy packets may have too much impact on the device life time, and so may be avoided. On the other hand, constrained nodes may be dedicated to specific applications, in which case, traffic pattern may expose the application or the type of node. For these nodes, not sending dummy packet may have some privacy implication that needs to be measured.

In some cases, devices are dedicated to a single application or a single transport protocol, in which case, the Next Header has a fix value.

Specific processing indications have not been standardized yet [I-D.nikander-esp-beet-mode] and is expected to result from an agreement between the peers. As a result, it is not expected to be part of a minimal implementation of ESP.

[RFC4303] mentions :

"The Next Header is a mandatory, 8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data. [...] the protocol value 59 (which means "no next header") MUST be used to designate a "dummy" packet. A transmitter MUST be capable of generating dummy packets marked with this value in the next protocol field, and a receiver MUST be prepared to discard such packets, without indicating an error."

7. ICV

The ICV depends on the crypto-suite used. Currently recommended [RFC8221] only recommend crypto-suites with an ICV which makes the ICV a mandatory field.

As detailed in Section 8 we recommend to use authentication, the ICV field is expected to be present that is to say with a size different from zero. This makes it a mandatory field which size is defined by the security recommendations only.

[RFC4303] mentions :

"The Integrity Check Value is a variable-length field computed over the ESP header, Payload, and ESP trailer fields. Implicit ESP trailer fields (integrity padding and high-order ESN bits, if applicable) are included in the ICV computation. The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a combined mode algorithm that uses an ICV. The length of the field is specified by the integrity algorithm selected and associated with the SA. The integrity algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation."

8. Cryptographic Suites

The cryptographic suites implemented are an important component of ESP. The recommended suites to use are expected to evolve over time and implementer SHOULD follow the recommendations provided by [RFC8221] and updates. Recommendations are provided for standard nodes as well as constrained nodes.

This section lists some of the criteria that may be considered. The list is not expected to be exhaustive and may also evolve overtime. As a result, the list is provided as indicative:

1. Security: Security is the criteria that should be considered first for the selection of cipher suites. The security of cipher

suites is expected to evolve over time, and it is of primary importance to follow up-to-date security guidances and recommendations. The chosen cipher suites MUST NOT be known vulnerable or weak (see [RFC8221] for outdated ciphers). ESP can be used to authenticate only or to encrypt the communication. In the later case, authenticated encryption must always be considered [RFC8221].

2. **Interoperability:** Interoperability considers the cipher suites shared with the other nodes. Note that it is not because a cipher suite is widely deployed that is secured. As a result, security SHOULD NOT be weakened for interoperability. [RFC8221] and successors consider the life cycle of cipher suites sufficiently long to provide interoperability. constrained devices may have limited interoperability requirements which makes possible to reduce the number of cipher suites to implement.
3. **Power Consumption and Cipher Suite Complexity:** Complexity of the cipher suite or the energy associated to it are especially considered when devices have limited resources or are using some batteries, in which case the battery determines the life of the device. The choice of a cryptographic function may consider re-using specific libraries or to take advantage of hardware acceleration provided by the device. For example if the device benefits from AES hardware modules and uses AES-CTR, it may prefer AUTH_AES-XCBC for its authentication. In addition, some devices may also embed radio modules with hardware acceleration for AES-CCM, in which case, this mode may be preferred.
4. **Power Consumption and Bandwidth Consumption:** Similarly to the cipher suite complexity, reducing the payload sent, may significantly reduce the energy consumption of the device. As a result, cipher suites with low overhead may be considered. To reduce the overall payload size one may for example:
 1. Use of counter-based ciphers without fixed block length (e.g. AES-CTR, or ChaCha20-Poly1305).
 2. Use of ciphers with capability of using implicit IVs [I-D.ietf-ipsecme-implicit-iv].
 3. Use of ciphers recommended for IoT [RFC8221].
 4. Avoid Padding by sending payload data which are aligned to the cipher block length - 2 for the ESP trailer.

9. IANA Considerations

There are no IANA consideration for this document.

10. Security Considerations

Security considerations are those of [RFC4303]. In addition, this document provided security recommendations and guidances over the implementation choices for each fields.

11. Acknowledgment

The authors would like to thank Daniel Palomares, Scott Fluhrer, Tero Kivinen, Valery Smyslov, Yoav Nir, Michael Richardson for their valuable comments.

12. References

12.1. Normative References

- [I-D.ietf-ipsecme-implicit-iv] Migault, D., Guggemos, T., and Y. Nir, "Implicit IV for Counter-based Ciphers in Encapsulating Security Payload (ESP)", draft-ietf-ipsecme-implicit-iv-04 (work in progress), May 2018.
- [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>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.

- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <<https://www.rfc-editor.org/info/rfc8221>>.

12.2. Informative References

- [I-D.mglt-ipsecme-diet-esp]
Migault, D., Guggemos, T., Bormann, C., and D. Schinazi, "ESP Header Compression and Diet-ESP", draft-mglt-ipsecme-diet-esp-06 (work in progress), May 2018.
- [I-D.mglt-ipsecme-ikev2-diet-esp-extension]
Migault, D. and T. Guggemos, "Internet Key Exchange version 2 (IKEv2) extension for the ESP Header Compression (EHC) Strategy", draft-mglt-ipsecme-ikev2-diet-esp-extension-00 (work in progress), October 2017.
- [I-D.nikander-esp-beet-mode]
Nikander, P. and J. Melen, "A Bound End-to-End Tunnel (BEET) mode for ESP", draft-nikander-esp-beet-mode-09 (work in progress), August 2008.

Appendix A. Document Change Log

- [RFC Editor: This section is to be removed before publication]
- 00: First version published.
- 01: Clarified description
- 02: Clarified description

Authors' Addresses

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Email: daniel.migault@ericsson.com

Tobias Guggemos
LMU Munich
MNM-Team
Oettingenstr. 67
80538 Munich, Bavaria
Germany

Email: guggemos@nm.ifi.lmu.de
URI: <http://www.nm.ifi.lmu.de/~guggemos>

Light-Weight Implementation Guidance (lwig)
Internet-Draft
Intended status: Informational
Expires: April 24, 2019

D. Migault
Ericsson
T. Guggemos
LMU Munich
October 21, 2018

Minimal ESP
draft-mglt-lwig-minimal-esp-07

Abstract

This document describes a minimal implementation of the IP Encapsulation Security Payload (ESP) defined in RFC 4303. Its purpose is to enable implementation of ESP with a minimal set of options to remain compatible with ESP as described in RFC 4303. A minimal version of ESP is not intended to become a replacement of the RFC 4303 ESP, but instead to enable a limited implementation to interoperate with implementations of RFC 4303 ESP.

This document describes what is required from RFC 4303 ESP as well as various ways to optimize compliance with RFC 4303 ESP.

This document does not update or modify RFC 4303, but provides a compact description of how to implement the minimal version of the protocol. If this document and RFC 4303 conflicts then RFC 4303 is the authoritative description.

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 April 24, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

1. Requirements notation

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. Introduction

ESP [RFC4303] is part of the IPsec suite protocol [RFC4301]. IPsec is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity) and limited traffic flow confidentiality.

Figure 1 describes an ESP Packet. Currently ESP is implemented in the kernel of major multi purpose Operating Systems (OS). The ESP and IPsec suite is usually implemented in a complete way to fit multiple purpose usage of these OS. However, completeness of the IPsec suite as well as multi purpose scope of these OS is often performed at the expense of resources, or a lack of performance. As a result, constraint devices are likely to have their own implementation of ESP optimized and adapted to their specificities. With the adoption of IPsec by IoT devices with minimal IKEv2 [RFC7815] and ESP Header Compression (EHC) with [I-D.mglt-ipsecme-diet-esp] or [I-D.mglt-ipsecme-ikev2-diet-esp-extension], it becomes crucial that ESP implementation designed for constraint devices remain interoperable with the standard ESP implementation to avoid a fragmented usage of ESP. This document describes the the minimal properties and ESP implementation needs to meet.

For each field of the ESP packet represented in Figure 1 this document provides recommendations and guidance for minimal implementations. The primary purpose of Minimal ESP is to remain

interoperable with other nodes implementing RFC 4303 ESP, while limiting the standard complexity of the implementation.

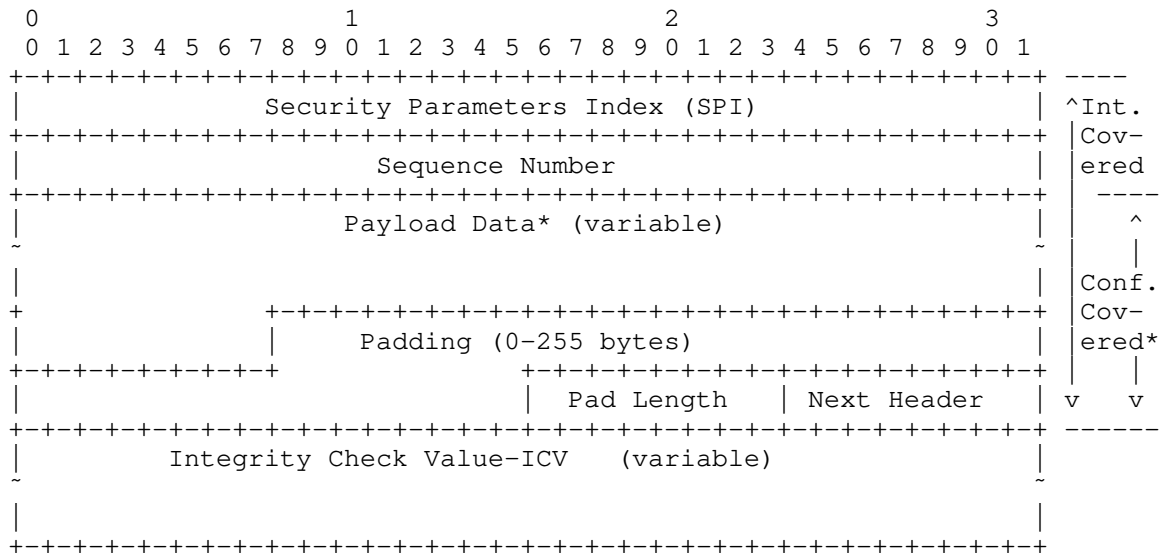


Figure 1: ESP Packet Description

3. Security Parameter Index (SPI) (32 bit)

According to the [RFC4303], the SPI is a mandatory 32 bits field and is not allowed to be removed.

The SPI has a local significance to index the Security Association (SA). From [RFC4301] section 4.1, nodes supporting only unicast communications can index their SA only using the SPI. On the other hand, nodes supporting multicast communications must also use the IP addresses and thus SA lookup needs to be performed using the longest match.

For nodes supporting only unicast communications, it is RECOMMENDED to index SA with the SPI only. Some other local constraints on the node may require a combination of the SPI as well as other parameters to index the SA.

It is RECOMMENDED to randomly generate the SPI indexing each inbound session. A random generation provides a stateless way to generate the SPIs, while keeping the probability of collision between SPIs relatively low. In case of collision, the SPI is simply re-generated.

However, for some constraint nodes, generating a random SPI may consume too much resource, in which case SPI can be generated using predictable functions or even a fix value. In fact, the SPI does not need to be random. Generating non random SPI MAY lead to privacy and security concerns. As a result, this alternative should be considered for devices that would be strongly impacted by the generation of a random SPI and after understanding the privacy and security impact of generating non random SPI.

When a constraint node uses fix value for SPIs, it imposes some limitations on the number of inbound SA. This limitation can be alleviated by how the SA lookup is performed. When fix SPI are used, it is RECOMMENDED the constraint node has as many SPI values as ESP session per host IP address, and that SA lookup includes the IP addresses.

Note that SPI value is used only for inbound traffic, as such the SPI negotiated with IKEv2 [RFC7296] or [RFC7815] by a peer, is the value used by the remote peer when it sends traffic. As SPI are only used for inbound traffic by the peer, this allows each peer to manage the set of SPIs used for its inbound traffic.

The use of fix SPI MUST NOT be considered as a way to avoid strong random generators. Such generator will be required in order to provide strong cryptographic protection and follow the randomness requirements for security described in [RFC4086]. Instead, the use of a fix SPI should only be considered as a way to overcome the resource limitations of the node, when this is feasible.

The use of a limited number of fix SPI or non random SPIs come with security or privacy drawbacks. Typically, a passive attacker may derive information such as the number of constraint devices connecting the remote peer, and in conjunction with data rate, the attacker may eventually determine the application the constraint device is associated to. If the SPI is fixed by a manufacturer or by some software application, the SPI may leak in an obvious way the type of sensor, the application involved or the model of the constraint device. When identification of the application or the hardware is associated to privacy, the SPI MUST be randomly generated. However, one needs to realize that in this case this is likely to be sufficient and a thorough privacy analysis is required. More specifically, traffic pattern MAY leak sufficient information in itself. In other words, privacy leakage is a complex and the use of random SPI is unlikely to be sufficient.

As the general recommendation is to randomly generate the SPI, constraint devices that will use a limited number of fix SPI are expected to be very constraint devices with very limited

capabilities, where the use of randomly generated SPI may prevent them to implement IPsec. In this case the ability to provision non random SPI enables these devices to secure their communications. These devices, due to there limitations, are expected to provide limited information and how the use of non random SPI impacts privacy requires further analysis. Typically temperature sensors, wind sensors, used outdoor do not leak privacy sensitive information. When used indoor, the privacy information is stored in the encrypted data and as such does not leak privacy.

As far as security is concerned, revealing the type of application or model of the constraint device could be used to identify the vulnerabilities the constraint device is subject to. This is especially sensitive for constraint devices where patches or software updates will be challenging to operate. As a result, these devices may remain vulnerable for relatively long period. In addition, predictable SPI enable an attacker to forge packets with a valid SPI. Such packet will not be rejected due to an SPI mismatch, but instead after the signature check which requires more resource and thus make DoS more efficient, especially for devices powered by batteries.

Values 0-255 SHOULD NOT be used. Values 1-255 are reserved and 0 is only allowed to be used internal and it MUST NOT be send on the wire.

[RFC4303] mentions :

"The SPI is an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound. The SPI field is mandatory. [...]"

"For a unicast SA, the SPI can be used by itself to specify an SA, or it may be used in conjunction with the IPsec protocol type (in this case ESP). Because the SPI value is generated by the receiver for a unicast SA, whether the value is sufficient to identify an SA by itself or whether it must be used in conjunction with the IPsec protocol value is a local matter. This mechanism for mapping inbound traffic to unicast SAs MUST be supported by all ESP implementations."

4. Sequence Number(SN) (32 bit)

According to [RFC4303], the Sequence Number (SN) is a mandatory 32 bits field in the packet.

The SN is set by the sender so the receiver can implement anti-replay protection. The SN is derived from any strictly increasing function that guarantees: if packet B is sent after packet A, then SN of packet B is strictly greater then the SN of packet A.

Some constraint devices may establish communication with specific devices, like a specific gateway, or nodes similar to them. As a result, the sender may know whereas the receiver implements anti-replay protection or not. Even though the sender may know the receiver does not implement anti replay protection, the sender **MUST** implement a always increasing function to generate the SN.

Usually, SN is generated by incrementing a counter for each packet sent. A constraint device may avoid maintaining this context and use another source that is known to always increase. Typically, constraint nodes using 802.15.4 Time Slotted Channel Hopping (TSCH), whose communication is heavily dependent on time, can take advantage of their clock to generate the SN. This would guarantee a strictly increasing function, and avoid storing any additional values or context related to the SN. When the use of a clock is considered, one should take care that packets associated to a given SA are not sent with the same time value.

For inbound traffic, it is **RECOMMENDED** to provide a anti-replay protection, and the size of the window depends on the ability of the network to deliver packet out of order. As a result, in environment where out of order packets is not possible the window size can be set to one. However, while **RECOMMENDED**, there is no requirements to implement an anti replay protection mechanism implemented by IPsec. A node **MAY** drop anti-replay protection provided by IPsec, and instead implement its own internal mechanism.

[RFC4303] mentions :

"This unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number. For a unicast SA or a single-sender multicast SA, the sender **MUST** increment this field for every transmitted packet. Sharing an SA among multiple senders is permitted, though generally not recommended. [...] The field is mandatory and **MUST** always be present even if the receiver does not elect to enable the anti-replay service for a specific SA."

5. Padding

The purpose of padding is to respect the 32 bit alignment of ESP. ESP **MUST** have at least one padding byte Pad Length that indicates the padding length. ESP padding bytes are generated by a succession of unsigned bytes starting with 1, 2, 3 with the last byte set to Pad Length, where Pad Length designates the length of the padding bytes.

Checking the padding structure is not mandatory, so the constraint device may not proceed to such checks, however, in order to

interoperate with existing ESP implementations, it MUST build the padding bytes as recommended by ESP.

In some situation the padding bytes may take a fix value. This would typically be the case when the Data Payload is of fix size.

[RFC4303] mentions :

"If Padding bytes are needed but the encryption algorithm does not specify the padding contents, then the following default processing MUST be used. The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, When this padding scheme is employed, the receiver SHOULD inspect the Padding field. (This scheme was selected because of its relative simplicity, ease of implementation in hardware, and because it offers limited protection against certain forms of "cut and paste" attacks in the absence of other integrity measures, if the receiver checks the padding values upon decryption.)"

ESP [RFC4303] also provides Traffic Flow Confidentiality (TFC) as a way to perform padding to hide traffic characteristics, which differs from respecting a 32 bit alignment. TFC is not mandatory and MUST be negotiated with the SA management protocol. TFC has not yet being widely adopted for standard ESP traffic. One possible reason is that it requires to shape the traffic according to one traffic pattern that needs to be maintained. This is likely to require extra processing as well as providing a "well recognized" traffic shape which could end up being counterproductive. As such TFC is not expected to be supported by a minimal ESP implementation.

As a result, TFC cannot not be enabled with minimal, and communication protection that were relying on TFC will be more sensitive to traffic shaping. This could expose the application as well as the devices used to a passive monitoring attacker. Such information could be used by the attacker in case a vulnerability is disclosed on the specific device. In addition, some application use - such as health applications - may also reveal important privacy oriented informations.

Some constraint nodes that have limited battery life time may also prefer avoiding sending extra padding bytes. However the same nodes may also be very specific to an application and device. As a result, they are also likely to be the main target for traffic shaping. In most cases, the payload carried by these nodes is quite small, and the standard padding mechanism may also be used as an alternative to TFC, with a sufficient trade off between the require energy to send

additional payload and the exposure to traffic shaping attacks. In addition, the information leaked by the traffic shaping may also be addressed by the application level. For example, it is preferred to have a sensor sending some information at regular time interval, rather when an specific event is happening. Typically a sensor monitoring the temperature, or a door is expected to send regularly the information - i.e. the temperature of the room or whether the door is closed or open) instead of only sending the information when the temperature has raised or when the door is being opened.

6. Next Header (8 bit)

According to [RFC4303], the Next Header is a mandatory 8 bits field in the packet. Next header is intended to specify the data contained in the payload as well as dummy packet. In addition, the Next Header may also carry an indication on how to process the packet [I-D.nikander-esp-beet-mode].

The ability to generate and receive dummy packet is required by [RFC4303]. For interoperability, it is RECOMMENDED a minimal ESP implementation discards dummy packets. Note that such recommendation only applies for nodes receiving packets, and that nodes designed to only send data may not implement this capability.

As the generation of dummy packets is subject to local management and based on a per-SA basis, a minimal ESP implementation may not generate such dummy packet. More especially, in constraint environment sending dummy packets may have too much impact on the device life time, and so may be avoided. On the other hand, constraint nodes may be dedicated to specific applications, in which case, traffic pattern may expose the application or the type of node. For these nodes, not sending dummy packet may have some privacy implication that needs to be measured. However, for the same reasons exposed in Section 5 traffic shaping at the IPsec layer may also introduce some traffic pattern, and on constraint devices the application is probably the most appropriated layer to limit the risk of leaking information by traffic shaping.

In some cases, devices are dedicated to a single application or a single transport protocol, in which case, the Next Header has a fix value.

Specific processing indications have not been standardized yet [I-D.nikander-esp-beet-mode] and is expected to result from an agreement between the peers. As a result, it is not expected to be part of a minimal implementation of ESP.

[RFC4303] mentions :

"The Next Header is a mandatory, 8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data. [...] the protocol value 59 (which means "no next header") MUST be used to designate a "dummy" packet. A transmitter MUST be capable of generating dummy packets marked with this value in the next protocol field, and a receiver MUST be prepared to discard such packets, without indicating an error."

7. ICV

The ICV depends on the crypto-suite used. Currently recommended [RFC8221] only recommend crypto-suites with an ICV which makes the ICV a mandatory field.

As detailed in Section 8 we recommend to use authentication, the ICV field is expected to be present that is to say with a size different from zero. This makes it a mandatory field which size is defined by the security recommendations only.

[RFC4303] mentions :

"The Integrity Check Value is a variable-length field computed over the ESP header, Payload, and ESP trailer fields. Implicit ESP trailer fields (integrity padding and high-order ESN bits, if applicable) are included in the ICV computation. The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a combined mode algorithm that uses an ICV. The length of the field is specified by the integrity algorithm selected and associated with the SA. The integrity algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation."

8. Cryptographic Suites

The cryptographic suites implemented are an important component of ESP. The recommended suites to use are expected to evolve over time and implementer SHOULD follow the recommendations provided by [RFC8221] and updates. Recommendations are provided for standard nodes as well as constraint nodes.

This section lists some of the criteria that may be considered. The list is not expected to be exhaustive and may also evolve overtime. As a result, the list is provided as indicative:

1. Security: Security is the criteria that should be considered first for the selection of cipher suites. The security of cipher

suites is expected to evolve over time, and it is of primary importance to follow up-to-date security guidances and recommendations. The chosen cipher suites MUST NOT be known vulnerable or weak (see [RFC8221] for outdated ciphers). ESP can be used to authenticate only or to encrypt the communication. In the later case, authenticated encryption must always be considered [RFC8221].

2. **Interoperability:** Interoperability considers the cipher suites shared with the other nodes. Note that it is not because a cipher suite is widely deployed that is secured. As a result, security SHOULD NOT be weakened for interoperability. [RFC8221] and successors consider the life cycle of cipher suites sufficiently long to provide interoperability. Constraint devices may have limited interoperability requirements which makes possible to reduce the number of cipher suites to implement.
3. **Power Consumption and Cipher Suite Complexity:** Complexity of the cipher suite or the energy associated to it are especially considered when devices have limited resources or are using some batteries, in which case the battery determines the life of the device. The choice of a cryptographic function may consider re-using specific libraries or to take advantage of hardware acceleration provided by the device. For example if the device benefits from AES hardware modules and uses AES-CTR, it may prefer AUTH_AES-XCBC for its authentication. In addition, some devices may also embed radio modules with hardware acceleration for AES-CCM, in which case, this mode may be preferred.
4. **Power Consumption and Bandwidth Consumption:** Similarly to the cipher suite complexity, reducing the payload sent, may significantly reduce the energy consumption of the device. As a result, cipher suites with low overhead may be considered. To reduce the overall payload size one may for example:
 1. Use of counter-based ciphers without fixed block length (e.g. AES-CTR, or ChaCha20-Poly1305).
 2. Use of ciphers with capability of using implicit IVs [I-D.ietf-ipsecme-implicit-iv].
 3. Use of ciphers recommended for IoT [RFC8221].
 4. Avoid Padding by sending payload data which are aligned to the cipher block length - 2 for the ESP trailer.

9. IANA Considerations

There are no IANA consideration for this document.

10. Security Considerations

Security considerations are those of [RFC4303]. In addition, this document provided security recommendations and guidances over the implementation choices for each fields.

11. Acknowledgment

The authors would like to thank Daniel Palomares, Scott Fluhrer, Tero Kivinen, Valery Smyslov, Yoav Nir, Michael Richardson for their valuable comments.

12. References

12.1. Normative References

- [I-D.ietf-ipsecme-implicit-iv]
Migault, D., Guggemos, T., and Y. Nir, "Implicit IV for Counter-based Ciphers in Encapsulating Security Payload (ESP)", draft-ietf-ipsecme-implicit-iv-05 (work in progress), June 2018.
- [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>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.

- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <<https://www.rfc-editor.org/info/rfc8221>>.

12.2. Informative References

- [I-D.mglt-ipsecme-diet-esp]
Migault, D., Guggemos, T., Bormann, C., and D. Schinazi, "ESP Header Compression and Diet-ESP", draft-mglt-ipsecme-diet-esp-06 (work in progress), May 2018.
- [I-D.mglt-ipsecme-ikev2-diet-esp-extension]
Migault, D., Guggemos, T., and D. Schinazi, "Internet Key Exchange version 2 (IKEv2) extension for the ESP Header Compression (EHC) Strategy", draft-mglt-ipsecme-ikev2-diet-esp-extension-01 (work in progress), June 2018.
- [I-D.nikander-esp-beet-mode]
Nikander, P. and J. Melen, "A Bound End-to-End Tunnel (BEET) mode for ESP", draft-nikander-esp-beet-mode-09 (work in progress), August 2008.

Appendix A. Document Change Log

[RFC Editor: This section is to be removed before publication]

-00: First version published.

-01: Clarified description

-02: Clarified description

Authors' Addresses

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Email: daniel.migault@ericsson.com

Tobias Guggemos
LMU Munich
MNM-Team
Oettingenstr. 67
80538 Munich, Bavaria
Germany

Email: guggemos@mn-m-team.org

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

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Fostering Code Reuse with New Elliptic Curves	2
2. Specification of Wei25519	3
3. Example Uses	3
3.1. ECDSA-SHA256-25519	3
3.2. Other Uses	4
4. Security Considerations	4
5. IANA Considerations	4
6. Normative References	4
Appendix A. Some (non-Binary) Elliptic Curves	5
A.1. Curves in short-Weierstrass Form	5
A.2. Montgomery Curves	5
A.3. Twisted Edwards Curves	6
Appendix B. Elliptic Curve Group Operations	6
B.1. Group Law for Weierstrass Curves	6
B.2. Group Law for Montgomery Curves	6
B.3. Group Law for Twisted Edwards Curves	7
Appendix C. Relationship Between Curve Models	8
C.1. Mapping between twisted Edwards Curves and Montgomery Curves	8
C.2. Mapping between Montgomery Curves and Weierstrass Curves	8
C.3. Mapping between twisted Edwards Curves and Weierstrass Curves	9
Appendix D. Curve25519 and Cousins	9
D.1. Curve Definition and Alternative Representations	9
D.2. Switching between Alternative Representations	10
D.3. Domain Parameters	11
Author's Address	13

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 $\text{GF}(q)$ 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 $\text{GF}(q)$. The points of $E_{\{a,d\}}$ are the ordered pairs (x, y) whose coordinates are elements of $\text{GF}(q)$ 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 $\text{GF}(q)$, whereas d is not, where the point $(0, 1)$ serves as the identity element. (Note that the identity element satisfies the defining equation.) See Appendix B.3 for details of the group operation. An Edwards curve is a twisted Edwards curve with $a=1$.

Appendix B. Elliptic Curve Group Operations

B.1. Group Law for Weierstrass Curves

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

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

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

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

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

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

B.2. Group Law for Montgomery Curves

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

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

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

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

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

$$x + 2x_1 = B \cdot \lambda^2 - A \text{ and } y + y_1 = \lambda(x_1 - x), \text{ where } \lambda = (3x_1^2 + 2Ax_1 + 1)/(2y_1).$$

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

B.3. Group Law for Twisted Edwards Curves

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

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

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

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

$$x = (x_1 y_2 + x_2 y_1) / (1 + d x_1 x_2 y_1 y_2) \text{ and } y = (y_1 y_2 - a x_1 x_2) / (1 - d x_1 x_2 y_1 y_2).$$

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

$$x = (2x_1 y_1) / (1 + d x_1^2 y_1^2) \text{ and } y = (y_1^2 - a x_1^2) / (1 - d x_1^2 y_1^2).$$

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

Appendix C. Relationship Between Curve Models

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

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

One can map points of the Montgomery curve $M_{\{A,B\}}$ to points of the twisted Edwards curve $E_{\{a,d\}}$, where $a := (A+2)/B$ and $d := (A-2)/B$ and, conversely, map points of the twisted Edwards curve $E_{\{a,d\}}$ to points of the Montgomery curve $M_{\{A,B\}}$, where $A := 2(a+d)/(a-d)$ and 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 $h \cdot n$, 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 $h_1 \cdot n_1$, where $h_1 = 4$ and where n_1 is a prime number. For this curve, the base point is defined to be the ordered pair (G_u, G_v) of elements of $GF(p)$, where $G_u = 9$ and where G_v 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 (G_x, G_y) 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 $\text{delta}=(p+A)/3$ is the element of $GF(p)$ defined by

```
delta 19298681539552699237261830834781317975544997444273427339909597
      334652188435537
```

```
(=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2
451)
```

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))
```

```
51042569399160536130206135233146329284152202253034631822681833788
666877215207
```

```
(=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 (G_x, G_y) of Edwards25519 corresponds to the base point (G'_x, G'_y) 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 $2^{255}-19$

(=0x7ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff)

h 8

n 72370055773322622139731865630429942408571163593799076060019509382
85454250989

(=2²⁵² + 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

(=370957059346694393431380835087545651895421138798432190163887855
33085940283555)

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

Gx 15112221349535400772501151409588531511454012693041857206046113283
949847762202

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

Gy 4/5

(=463168356949264781694283940034751631413079938662562256157830336
03165251855960)

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

Weierstrass curve-specific parameters (for Wei25519):

a 19298681539552699237261830834781317975544997444273427339909597334
573241639236

(=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
aaaaaaaa98 4914a144)

b 55751746669818908907645289078257140818241103727901012315294400837
956729358436

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

Gx' 19298681539552699237261830834781317975544997444273427339909597334
652188435546

(=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
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

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Fostering Code Reuse with New Elliptic Curves	3
2. Specification of Wei25519	3
3. Example Uses	3
3.1. ECDSA-SHA256-25519	3
3.2. Other Uses	4
4. Security Considerations	4
5. IANA Considerations	4
6. Normative References	4
Appendix A. Some (non-Binary) Elliptic Curves	6
A.1. Curves in short-Weierstrass Form	6
A.2. Montgomery Curves	6
A.3. Twisted Edwards Curves	6
Appendix B. Elliptic Curve Group Operations	7
B.1. Group Law for Weierstrass Curves	7
B.2. Group Law for Montgomery Curves	7
B.3. Group Law for Twisted Edwards Curves	8
Appendix C. Relationship Between Curve Models	8
C.1. Mapping between twisted Edwards Curves and Montgomery Curves	8
C.2. Mapping between Montgomery Curves and Weierstrass Curves	9
C.3. Mapping between twisted Edwards Curves and Weierstrass Curves	10
Appendix D. Curve25519 and Cousins	10
D.1. Curve Definition and Alternative Representations	10
D.2. Switching between Alternative Representations	10
D.3. Domain Parameters	12
Appendix E. Further Mappings	14
E.1. Isomorphic Mapping between Weierstrass Curves	14
E.2. Isogeneous Mapping between Weierstrass Curves	15
Appendix F. Further Cousins of Curve25519	15
F.1. Further Alternative Representations	15
F.2. Further Switching	15
F.3. Further Domain Parameters	16
Author's Address	17

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

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

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

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

B.2. Group Law for Montgomery Curves

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

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

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

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

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

$$x + 2x_1 = B\lambda^2 - A \text{ and } y + y_1 = \lambda(x_1 - x), \text{ where } \lambda = (3x_1^2 + 2Ax_1 + 1)/(2y_1).$$

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

B.3. Group Law for Twisted Edwards Curves

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

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

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

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

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

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

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

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

Appendix C. Relationship Between Curve Models

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

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

One can map points of the Montgomery curve $M_{\{A,B\}}$ to points of the twisted Edwards curve $E_{\{a,d\}}$, where $a:=(A+2)/B$ and $d:=(A-2)/B$ and, conversely, map points of the twisted Edwards curve $E_{\{a,d\}}$ to points of the Montgomery curve $M_{\{A,B\}}$, where $A:=2(a+d)/(a-d)$ and 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 $h \cdot n$, 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 $h_1 \cdot n_1$, where $h_1=4$ and where n_1 is a prime number. For this curve, the base point is the point (G_u, G_v) , where $G_u=9$ and where G_v 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 (G_x, G_y) , 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 (G_x', G_y') , 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 (G_u, G_v) of Curve25519 corresponds to the base point (G_x', G_y') 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 $\text{delta}=(p+A)/3$ is the element of $\text{GF}(p)$ defined by

```
delta 19298681539552699237261830834781317975544997444273427339909597
334652188435537
```

```
(=0x2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaad2
451)
```

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

```
c  sqrt(-(A+2))
```

```
51042569399160536130206135233146329284152202253034631822681833788
666877215207
```

```
(=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 (G_x, G_y) of Edwards25519 corresponds to the base point (G_x', G_y') 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 $(c \cdot (3 \cdot x - A) / (3 \cdot y), (3 \cdot x - A - 3) / (3 \cdot x - A + 3))$.

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

    (=0x7fffffffff ffffffffff ffffffffff ffffffffff ffffffffff ffffffffff
    ffffffffff ffffffff)

h   8

n  72370055773322622139731865630429942408571163593799076060019509382
    85454250989

    (=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

(=370957059346694393431380835087545651895421138798432190163887855
33085940283555)

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

Gx 15112221349535400772501151409588531511454012693041857206046113283
949847762202

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

Gy 4/5

(=463168356949264781694283940034751631413079938662562256157830336
03165251855960)

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

Weierstrass curve-specific parameters (for Wei25519):

a 19298681539552699237261830834781317975544997444273427339909597334
573241639236

(=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
aaaaaaaa98 4914a144)

b 55751746669818908907645289078257140818241103727901012315294400837
956729358436

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

Gx' 19298681539552699237261830834781317975544997444273427339909597334
652188435546

(=0x2aaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa
aaaaaaaa aad245a)

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 $a' \equiv -3 \pmod{p}$ and one uses so-called Jacobian coordinates with a particular projective version of the addition laws of Appendix B.1. While not all Weierstrass curves can be put into this form, all traditional NIST curves have domain parameter $a \equiv -3$, 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 $a' := -3 \pmod{p}$, even if a'/a is not a fourth power in $GF(q)$. In that case, this mapping cannot be an isomorphism (see Appendix E.1) and, thereby, does not define a one-to-one correspondence. Instead, the mapping is a so-called isogeny (or homomorphism). Since most elliptic curve operations process points of prime order or use so-called "co-factor multiplication", in practice the resulting mapping has similar properties. In particular, one can still take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with domain parameter a unequal to $-3 \pmod{p}$ on a corresponding isogenous Weierstrass curve with domain parameter $a' = -3 \pmod{p}$ 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    47731687248873559672555216906496754195083410699918207029391079363
    6321486119

    (=0x10e26daca93602704c7e6cff9efe595764cb5c9e04931f6fdeefc657d4e5
    27),
```

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    45793404337388339159414415854563976158160282736335993851976016290
    777777599260

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

G1x' 218726072268944427441327971914352883414836203960572472224621495
    35754145422686

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

G1y' 139436179034864291344077235766386796155987755307479919871866321
    47013341290929

    (=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
```

(=0x7ffffffff 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