

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 10, 2013

J. Arkko  
A. Eriksson  
A. Keranen  
Ericsson  
July 9, 2012

Building Power-Efficient CoAP Devices for Cellular Networks  
draft-arkko-core-cellular-00

Abstract

This memo discusses the use of the Constrained Application Protocol (CoAP) protocol in building sensors and other devices that employ cellular networks as a communications medium. Building communicating devices that employ these networks is obviously well known, but this memo focuses specifically on techniques necessary to minimize power consumption.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

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

## Table of Contents

1. Introduction . . . . .	3
2. Goals for Low-Power Operation . . . . .	4
3. Link-Layer Assumptions . . . . .	7
4. Scenarios . . . . .	9
5. Discovery and Registration . . . . .	9
6. Data Formats . . . . .	11
7. Real-Time Reachable Devices . . . . .	11
8. Sleepy Devices . . . . .	12
8.1. Implementation Considerations . . . . .	13
9. Security Considerations . . . . .	14
10. IANA Considerations . . . . .	14
11. References . . . . .	15
11.1. Normative References . . . . .	15
11.2. Informative References . . . . .	15
Appendix A. Acknowledgments . . . . .	16
Authors' Addresses . . . . .	16

## 1. Introduction

This memo discusses the use of the Constrained Application Protocol (CoAP) protocol [I-D.ietf-core-coap] in building sensors and other devices that employ cellular networks as a communications medium. Building communicating devices that employ these networks is obviously well known, but this memo focuses specifically on techniques necessary to minimize power consumption. CoAP has many advantages, including being simple to implement; a thousand lines for the entire software above IP layer is plenty for a CoAP-based sensor, for instance. However, while many of these advantages are obvious and easily obtained, optimizing power consumption remains challenging and requires careful design [I-D.arkko-core-sleepy-sensors].

The memo targets primarily 3GPP cellular networks in their 2G, 3G, and LTE variants and their future enhancements, including possible power efficiency improvements at the radio and link layers. The exact standards or details of the link layer or radios are not relevant for our purposes, however. To be more precise, the material in this memo is suitable for any large-scale, public network that employs point-to-point communications model and radio technology.

Our focus is devices that need to be optimized for power usage, and on devices that employ CoAP. As a general technology, CoAP is similar to HTTP. It can be used in various ways and network entities may take on different roles. This freedom allows the technology to be used in efficient and less efficient ways. Some guidance is needed to understand what communication models over CoAP are recommended when low power usage is a critical goal.

The recommendations in this memo should be taken as complementary to device hardware optimization, microelectronics improvements, and further evolution of the underlying link and radio layers. Further gains in power efficiency can certainly be gained on several fronts; the approach that we take in this memo is to do what can be done at the IP, transport, and application layers to provide the best possible power efficiency. Application implementors generally have to use the current generation microelectronics, currently available radio networks and standards, and so on. This focus in our memo should by no means be taken as an indication that further evolution in these other areas is unnecessary. Such evolution is useful, is ongoing, and is generally complementary to the techniques presented in this memo. The evolution of underlying technologies may change what techniques described here are useful for a particular application, however.

The rest of this memo is structured as follows. Section 2 discusses the need and goals for low-power devices. Section 3 outlines our

expectations for the low layer communications model. Section 4 describes the two scenarios that we address, and Section 5, Section 6, Section 7 and Section 8 give guidelines for use of CoAP in these scenarios.

## 2. Goals for Low-Power Operation

There are many situations where power usage optimization is unnecessary. Optimization may not be necessary on devices that can run on power feed over wired communications media, such as in Power-over-Ethernet (PoE) solutions. These devices may require a rudimentary level of power optimization techniques just to avoid keep overall energy costs and aggregate power feed sizes at a reasonable level, but more extreme techniques necessary for battery powered devices are not required. The situation is similar with devices that can be easily be connected to mains power. Other types of devices may get an occasional charge of power from energy harvesting techniques. For instance, some environmental sensors can run on solar cells. Typically, these devices still have to regulate their power usage in a strict manner, for instance to be able to use as small and inexpensive solar cells as possible.

In battery operated devices the power usage is even more important. For instance, one of the authors employs over a hundred different sensor devices in his home network. A majority of these devices are wired and run on PoE, but in most environments this would be impractical because the necessary wires do not exist. The future is in wireless solutions that can cover buildings and other environments without assuming a pre-existing wired infrastructure. In addition, in many cases it is impractical to provide a mains power source. Often there are no power sockets easily available in the locations that the devices need to be in, and even if there were, setting up the wires and power adapters would be more complicated than installing a standalone device without any wires.

Yet, with a large number of devices the battery lifetimes become critical. Cost and practical limits dictate that devices can be largely just bought and left on their own. For instance, with hundred devices, even a ten-year battery lifetime results in a monthly battery change for one device within the network. This may be impractical in many environments. In addition, some devices may be physically difficult to reach for a battery change. Or, a large group of devices -- such as utility meters or environmental sensors -- cannot be economically serviced too often, even if in theory the batteries could be changed.

POWER SOURCE	SENSOR COMMUNICATION INTERVAL		
	Seconds	Minutes or Hours	Days and longer
Battery	Low-power	Low-power or Always-off	Always-off
Harvesting	Low-power	Low-power or Always-off	Always-off
Mains	Always-on	Always-on	Always-on

Figure 1: Power usage strategies for different classes of applications

Many of these situations lead to a requirement for minimizing power usage and/or maximizing battery lifetimes. A summary of the different situations for sensor-type devices is shown in Figure 1 above. Unfortunately, much of our current technology has been built with different objectives in mind. Networked devices that are "always on", gadgets that require humans to recharge it every couple of days, and protocols that have been optimized to maximize throughput rather than conserve resources.

Long battery lifetimes are required for many applications, however. In some cases these lifetimes should be in the order of years or even a decade or longer. Some communication devices already reach multi-year lifetimes, and continuous improvement in low-power electronics and advances in radio technology keep pushing these lifetimes longer. However, it is perhaps fair to say that battery lifetimes are generally too short at present time.

The general strategies for power usage from Figure 1 can be described as follows:

#### Always-on

Under this strategy, 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 a 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 in the network all the time.

### Always-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 is connected to the network during those times. These devices re-attach to the network as they are woken up. The main optimization goal is to minimize the effort during such re-attachment process and any resulting application communications.

If the device sleeps for long periods of time, the relative increase in energy expenditure during reattachment for infrequent communication may be acceptable.

### Low-power

These devices need to operate on very small amount of power, but still be able to communicate in a relatively frequent basis. This implies that extremely low power solutions needs 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 network 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, tuning the communications frequency, and paging frequency and other parameters appropriately.

Power usage can not be evaluated solely based on lower layer communications. The entire system, including upper layer protocols and applications is responsible for the power consumption as a whole. The lower communication layers have already adopted many techniques that can be used to reduce power usage, such as scheduling device wake-up times. Further reductions will likely need some co-operation from the upper layers so that unnecessary communications, denial-of-service attacks on power consumptions, and other power drains are eliminated.

Of course, application requirements ultimately determine what kinds of communications are necessary. For instance, some applications require more data to be sent than others. The purpose of the guidelines in this memo is not to prefer one or the other application, but to provide guidance on how to minimize the amount of communications overhead that is not directly required by the application. While such optimization is generally useful, it is relatively speaking most noticeable in applications that transfer only a small amount of data, or operate only infrequently.

### 3. Link-Layer Assumptions

We assume that the underlying communications network can be any large-scale, public network that employs point-to-point communications model and radio technology. 2G, 3G, and LTE networks are examples of such networks, but not the only possible networks with these characteristics.

In the following we look at some of these characteristics and their implications. Note that in most cases these characteristics are not properties of the specific networks but rather inherent in the concept of public networks.

#### Public networks

Using a public network service implies that applications can be deployed without having to build a network to go with them. For economical reasons, only the largest users (such as utility companies) could afford to build their own network, and even they would not be able to provide a world-wide coverage. This means that applications where coverage is important can be built. For instance, most transport sector applications require national or even world-wide coverage to work.

But there are other implications, as well. By definition, the network is not tailored for this application and with some exceptions, the traffic passes through the Internet. One implication of this is that there are generally no application-specific network configurations or discovery support. For instance, the public network helps devices to get on the Internet, set up default routers, configure DNS servers, and so on, but does nothing for configuring possible higher-layer functions, such as servers the device might need to contact to perform its application functions.

Public networks often provide web proxies, and these can in some cases make a significant improvement for delays and cost of communication over the wireless link. For instance, collecting content from a large number of servers used to render a web page and resolving their DNS names in a proxy instead of the user's device may cut down on the general chattiness of the communications, therefore reducing overall delay in completing the entire transaction. However, as of today such proxies are provided only for HTTP communications, not for CoAP.

Similarly, given the lack of available IPv4 addresses, the chances are that many devices are behind a network address translation (NAT) device. This means that they are not easily reachable as

servers. Alternatively, the devices may be directly on the global Internet (either on IPv4 or IPv6) and easily reachable as servers. Unfortunately, this may mean that they also receive unwanted traffic, which may have implications for both power consumption and service costs.

#### Point-to-point link model

This is a common link model in cellular networks. One implication of this model is that there will be no other nodes on the same link, except maybe for the service provider's router. As a result, multicast discovery can not be reasonably used for any local discovery purposes. While the configuration of the service provider's router for specific users is theoretically possible, in practice this is difficult to achieve, at least for any small user that can not afford a network-wide contract for a private APN. The public network access service has little per-user tailoring.

#### Radio technology

The use of radio technology means that power is needed to operate the radios. Transmission generally requires more power than reception. However, radio protocols have generally been designed so that a device checks periodically whether it has messages. In a situation where messages arrive seldomly or not at all, this checking consumes energy. Research has shown that these periodic checks (such as LTE paging message reception) are often a far bigger contributor to energy consumption than message transmission.

Note that for situations where there are several applications on the same device wishing to communicate with the Internet in some manner, bundling those applications together at the same time can be very useful. Some guidance for these techniques in the smartphone context can be found in [Android-Bundle].

Naturally, each device has a freedom to decide when it sends messages. In addition, we assume that there is some way for the devices to control when or how often it wants to receive messages. Specific methods for doing this depend on the specific network being used and also tend to change as improvements in the design of these networks are incorporated. The reception control methods generally come in two variants, fine grained mechanisms that deal with how often the device needs to wake-up for paging messages, and more crude mechanisms where the device simply disconnects from the network for a period of time. There are associated costs and benefits to each method, but those are not relevant for this memo, as long as some control method exists.

#### 4. Scenarios

Not all applications or situations are equal. They may require different solutions or communication models. This memo focuses on two common scenarios:

##### Real-Time Reachable Devices

This scenario involves all communication that requires real-time or near real-time communications with a device. That is, a network entity must be able to reach the device with a small time lag at any time, and no pre-agreed wake-up schedule can be arranged. By "real-time" we mean any reasonable end-to-end communications latency, be it measured in milliseconds or seconds. However, unpredictable sleep states are not expected.

Examples of devices in this category include sensors that must be measurable from a remote source at any instant in time, such as process automation sensors and actuators that require immediate action, such as lightbulbs or door locks.

##### Sleepy Devices

This scenario involves freedom to choose when device communicates. The device is often expected to be able to be in a sleep state for much of its time. The device itself can choose when it communicates, or it lets the network assist in this task.

Examples of devices in this category include sensors that track slowly changing values, such as temperature sensors and actuators that control a relatively slow process, such as heating systems.

Note that there may be hard real-time requirements, but they are expressed in terms of how fast the device can communicate, not in terms of how fast it can respond to a network stimuli. For instance, a fire detector can be classified as a sleepy device as long as it can internally quickly wake up on detecting fire and initiate the necessary communications without delay.

#### 5. Discovery and Registration

In both scenarios the device will be attached to a public network. Without special arrangements, the device will also get a dynamically assigned IP address or an IPv6 prefix. At least one but typically several router hops separate the device from its communicating peers such as application servers. As a result, the address or even the existence of the device is typically not immediately obvious to the

other nodes participating in the application. As discussed earlier, multicast discovery has limited value in public networks; network nodes cannot practically discover individual devices in a large public network. And the devices can not discover who they need to talk, as the public network offers just basic Internet connectivity.

Our recommendation is to initiate a discovery and registration process. This allows each device to inform its peers that it has connected to the network and that it is reachable at a given IP address.

The registration part is easy; a resource directory or mirror proxy can be used. The device should perform the necessary registration with these devices, for instance, as specified in [I-D.shelby-core-resource-directory] and [I-D.vial-core-mirror-proxy]. In order to do this registration, the device needs to know its CORE Link Format description, as specified in [I-D.ietf-core-link-format]. In essence, the registration process involves performing a GET on `.well-known/core/?rt=core-rd` at the address of the resource directory (or `rt=core-mp` for mirror proxies), and then doing a POST on the path of the discovered resource.

However, current CoAP specifications provide limited support for discovering the resource directory or mirror proxy. Local multicast discovery only works in LAN-type networks, but not in these public cellular networks. Our recommended alternate methods for discovery are the following:

#### Manual Configuration

The DNS name of the resource directory or mirror proxy is manually configured. This approach is suitable in situations where the owner of the devices has the resources and capabilities to do the configuration. For instance, a utility company can typically program its metering devices to point to the company servers.

#### Manufacturer Server

The DNS name of the directory or proxy is hardwired to the software by the manufacturer, and the directory or proxy is actually run by the manufacturer. This approach is suitable in many consumer usage scenarios, where it would be unreasonable to assume that the consumer runs any specific network services. The manufacturer's web interface and the directory/proxy servers can co-operate to provide the desired functionality to the end user. For instance, the end user can register a device identity in the manufacturer's web interface and ask specific actions to be taken when the device does something.

### Delegating Manufacturer Server

The DNS name of the directory or proxy is hardwired to the software by the manufacturer, but this directory or proxy merely redirects the request to a directory or proxy run by the whoever bought the device. This approach is suitable in many enterprise environments, as it allows the enterprise to be in charge of actual data collection and device registries; only the initial bootstrap goes through the manufacturer. In many cases there are even legal requirements (such as EU privacy laws) that prevent providing unnecessary information to third parties.

### Common Global Resolution Infrastructure

The delegating manufacturer server model could be generalized into a reverse-DNS -like discovery infrastructure that could answer the question "this is device with identity ID, where is my home registration server?". However, at present no such resolution system exists. (Note: The EPCGlobal system for RFID resolution is reminiscent of this approach.)

## 6. Data Formats

A variety of data formats exist for passing around data. These data formats include XML, JSON, EXI, and text formats. Message lengths can have a significant effect on the amount of energy required for the communications, and such it is highly desirable to keep message lengths minimal. At the same time, extreme optimization can affect flexibility and ease of programming. The authors recommend [I-D.jennings-senml] as a compact, yet easily processed and extendable textual format.

## 7. Real-Time Reachable Devices

These devices are often best modeled as CoAP servers. The device will have limited control on when it receives messages, and it will have to listen actively for messages, up to the limits of the underlying link layer. If the device acts also in client role in some phase of its operation, it can control how many transmissions it makes on its own behalf.

The packet reception checks should be tailored according to the requirements of the application. If sub-second response time is not needed, a slightly more infrequent checking process may save some power.

For sensor-type devices, the CoAP OBSERVE extension [I-D.ietf-core-observe] may be supported. This allows the sensor to track changes to the sensed value, and make an immediate observation response upon a change. This may reduce the amount of polling needed to be done by the client. Unfortunately, it does not reduce the time that the device needs to be listening for requests. Subscription requests from other clients than the currently registered one may come at any time, the current client may change its request, and the device still needs to respond to normal queries as a server. As a result, the sensor can not rely having to communicate only on its own choice of observation interval.

In order to act as a server, the device needs to be placed in a public IPv4 address, be reachable over IPv6, or hosted in a private network. If the the device is hosted on a private network, then all other nodes need to access this device also need to reside in the same private network. There are multiple ways to provide private networks over public cellular networks. One approach is to dedicate a special Access Point Name or APN for the private network. Corporate access via cellular networks has often been arranged in this manner, for instance. Another approach is to use Virtual Private Networking (VPN) technology, for instance IPsec-based VPNs.

Power consumption from unwanted traffic is problematic in these devices, unless placed in a private network or protected by a operator-provided firewall service. Devices on an IPv6 network will have some protection through the nature of the  $2^{64}$  address allocation for a single terminal in a 3GPP cellular network; the attackers will be unable to guess the full IP address of the device. However, this protects only the device from processing a packet, but since the network will still deliver the packet to any of the addresses within the assigned 64-bit prefix, packet reception costs are still incurred.

Note that the the VPN approach can not prevent unwanted traffic received at the tunnel endpoint address, and may require keep-alive traffic. Special APNs can solve this issue, but require explicit arrangement with the service provider.

## 8. Sleepy Devices

These devices are best modeled as devices that can delegate queries to some other node. For instance, as mirror proxy clients [I-D.vial-core-mirror-proxy]. When the device initializes itself, it makes a registration of itself in a mirror proxy as described above in Section 5 and then continues to send periodic updates of sensor values.

As a result, the device acts only as a client, not a server, and can shut down all communication channels while it is during its sleeping period. The length of the sleeping period depends on power and application requirements. Some environmental sensors might use a day or a week as the period, while other devices may use a smaller values ranging from minutes to hours.

Other approaches for delegation include CoAP-options described in [I-D.castellani-core-alive] [I-D.fossati-core-publish-monitor-options]. In this memo we use mirror proxies as an example, because of their ability to work with both HTTP and CoAP implementations; but the concepts are similar and the IETF work is still in progress so the final protocol details are yet to be decided.

The ability to shut down communications and act as only a client has four impacts:

- o Radio transmission and reception can be turned off during the sleeping period, reducing power consumption significantly.
- o However, some power and time is consumed by having to re-attach to the network after the end of a sleep period.
- o The window of opportunity for unwanted traffic to arrive is much smaller, as the device is listening for traffic only part of the time. Note that networks may cache packets for some time though. On the other hand, stateful firewalls can effectively remove much of unwanted traffic for client type devices.
- o The device may exist behind a NAT or a firewall without being impacted. Note that "Simple Security" basic IPv6 firewall capability [RFC6092] blocks inbound UDP traffic by default, so just moving to IPv6 is not direct solution to this problem.

For sleepy devices that represent actuators, it is also possible to use the mirror proxy model. The device can make periodic polls to the proxy to determine if a variable has changed.

### 8.1. Implementation Considerations

There are several challenges in implementing sleepy devices. They need hardware that can be put to an appropriate sleep mode but yet awakened when it is time to do something again. This is not always easy in all hardware platforms. It is important to be able to shut down as much of the hardware as possible, preferably down to everything else except a clock circuit. The platform also needs to support re-awakening at suitable time scales, as otherwise the device

needs to be powered up too frequently.

Most commercial cellular modem platforms do not allow applications to suspend the state of the communications stack. Hence, after a power-off period they need to re-establish communications, which takes some amount of time and extra energy.

Implementations should have a coordinated understanding of the state and sleeping schedule. For instance, it makes no sense to keep a CPU powered up, waiting for a message when the lower layer has been told that the next possible paging opportunity is some time away.

The cellular networks have a number of adjustable configuration parameters, such as the maximum used paging interval. Proper setting of these values has an impact on the power consumption of the device, but with the current business practices, such settings are rarely negotiated when the user's subscription is provisioned.

## 9. Security Considerations

There are no particular security aspects with what has been discussed in this memo, except for the ability to delegate queries for a resource to another node. Depending on how this is done, there are obvious security issues which have largely NOT yet been addressed in the relevant Internet Drafts [I-D.vial-core-mirror-proxy] [I-D.castellani-core-alive] [I-D.fossati-core-publish-monitor-options]. However, we point out that in general, security issues in delegation can be solved either through reliance on your local network support nodes (which may be quite reasonable in many environments) or explicit end-to-end security. Explicit end-to-end security through nodes that are awake at different times means in practice end-to-end data object security. We have implemented one such mechanism for sleepy nodes as described in [I-D.aks-crypto-sensors].

The security considerations relating to CoAP [I-D.ietf-core-coap] and the relevant link layers should apply. Note that cellular networks universally employ per-device authentication, integrity protection, and for most of the world, encryption of all their communications. Additional protection of transport sessions is possible through mechanisms described in [I-D.ietf-core-coap] or data objects.

## 10. IANA Considerations

There are no IANA impacts in this memo.

## 11. References

### 11.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-06 (work in progress), May 2011.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.vial-core-mirror-proxy]  
Vial, M., "CoRE Mirror Proxy",  
draft-vial-core-mirror-proxy-00 (work in progress),  
March 2012.
- [I-D.shelby-core-resource-directory]  
Shelby, Z. and S. Krco, "CoRE Resource Directory",  
draft-shelby-core-resource-directory-00 (work in  
progress), June 2011.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-11 (work in progress),  
January 2012.
- [I-D.jennings-senml]  
Jennings, C., "Media Type for Sensor Markup Language  
(SENML)", draft-jennings-senml-05 (work in progress),  
March 2011.

### 11.2. Informative References

- [RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in  
Customer Premises Equipment (CPE) for Providing  
Residential IPv6 Internet Service", RFC 6092,  
January 2011.
- [I-D.arkko-core-sleepy-sensors]  
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O.  
Novo, "Implementing Tiny COAP Sensors",  
draft-arkko-core-sleepy-sensors-01 (work in progress),  
July 2011.
- [I-D.arkko-core-security-arch]  
Arkko, J. and A. Keranen, "CoAP Security Architecture",

draft-arkko-core-security-arch-00 (work in progress),  
July 2011.

[I-D.aks-crypto-sensors]

Sethi, M., Arkko, J., Keranen, A., and H. Rissanen,  
"Practical Considerations and Implementation Experiences  
in Securing Smart Object Networks",  
draft-aks-crypto-sensors-02 (work in progress),  
March 2012.

[I-D.ietf-jose-json-web-signature]

Jones, M., Bradley, J., and N. Sakimura, "JSON Web  
Signature (JWS)", draft-ietf-jose-json-web-signature-01  
(work in progress), March 2012.

[I-D.castellani-core-alive]

Castellani, A. and S. Loreto, "CoAP Alive Message",  
draft-castellani-core-alive-00 (work in progress),  
March 2012.

[I-D.fossati-core-publish-monitor-options]

Fossati, T., Giacomini, P., and S. Loreto, "Publish and  
Monitor Options for CoAP",  
draft-fossati-core-publish-monitor-options-01 (work in  
progress), March 2012.

[Android-Bundle]

Developer.Android.Com, "Optimizing Downloads for Efficient  
Network Access", Android developer note [http://  
developer.android.com/training/efficient-downloads/  
efficient-network-access.html](http://developer.android.com/training/efficient-downloads/efficient-network-access.html), <[http://  
developer.android.com/training/efficient-downloads/  
efficient-network-access.html](http://developer.android.com/training/efficient-downloads/efficient-network-access.html)>.

## Appendix A. Acknowledgments

The authors would like to thank Zach Shelby, Jan Holler, Salvatore Loreto, Matthew Vial, Thomas Fossati, Mohit Sethi, Jan Melen, Joachim Sachs, Heidi-Maria Rissanen, Sebastien Pierrel, Kumar Balachandran, Muhammad Waqas Mir, Cullen Jennings, Markus Isomaki, Hannes Tschofenig, and Anna Larmo for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net

Anders Eriksson  
Ericsson  
Stockholm 164 83  
Sweden

Email: anders.e.eriksson@ericsson.com

Ari Keranen  
Ericsson  
Jorvas 02420  
Finland

Email: ari.keranen@ericsson.com



LWIG Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2013

C. Bormann, Ed.  
Universitaet Bremen TZI  
February 25, 2013

Guidance for Light-Weight Implementations of the Internet Protocol Suite  
draft-ietf-lwig-guidance-03

Abstract

Implementation of Internet protocols on small devices benefits from light-weight implementation techniques, which are often not documented in an accessible way.

This document provides a first outline of and some initial content for the Light-Weight Implementation Guidance document planned by the IETF working group LWIG.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Objectives . . . . .	4
1.2. Call for contributions . . . . .	5
1.3. Terminology used in this document . . . . .	5
1.4. Scope of the present document . . . . .	6
1.5. Terminology boilerplate . . . . .	6
2. Drawing the Landscape . . . . .	6
2.1. Design Objectives . . . . .	6
2.2. Implementation Styles . . . . .	7
2.3. Roles of nodes . . . . .	8
2.4. Overview over the document . . . . .	8
3. Data Plane Protocols . . . . .	8
3.1. Link Adaptation Layer . . . . .	8
3.1.1. Fragmentation in a 6LoWPAN Route-Over Configuration . . . . .	8
3.1.1.1. Implementation Considerations for Not-So-Constrained Nodes . . . . .	10
3.2. Network Layer . . . . .	10
3.3. Transport Layer . . . . .	10
3.3.1. TCP . . . . .	10
3.3.1.1. Absolutely required TCP behaviors for proper functioning and interoperability . . . . .	11
3.3.1.2. Strongly encouraged, but non-essential, behaviors of TCP . . . . .	12
3.3.1.3. Experimental extensions that are not yet standard practices . . . . .	13
3.3.1.4. Others . . . . .	13
3.4. Application Layer . . . . .	14
3.4.1. General considerations about Application Programming Interfaces (APIs) . . . . .	14
3.4.2. Constrained Application Protocol (CoAP) . . . . .	14
3.4.2.1. Message Layer Processing . . . . .	15
3.4.2.2. Message Parsing . . . . .	16
3.4.2.3. Storing Used Message IDs . . . . .	17
3.4.3. (Other Application Protocols...) . . . . .	20
4. Control Plane Protocols . . . . .	20
4.1. Link Layer Support . . . . .	20
4.2. Network Layer . . . . .	20
4.3. Routing . . . . .	21
4.4. Host Configuration and Lookup Services . . . . .	21
4.5. Network Management . . . . .	21
4.5.1. SNMP . . . . .	21
4.5.1.1. Background . . . . .	21
4.5.1.2. Revisiting SNMP implementation for resource	

constrained devices . . . . .	22
4.5.1.3. Proposed approach for building an memory efficient SNMP agent . . . . .	22
4.5.1.4. Example . . . . .	23
4.5.1.5. Further improvements . . . . .	25
4.5.1.6. Conclusion . . . . .	26
5. Security protocols . . . . .	26
5.1. Cryptography for Constrained Devices . . . . .	26
5.2. Transport Layer Security . . . . .	26
5.3. Network Layer Security . . . . .	26
5.4. Network Access Control . . . . .	26
5.4.1. PANA . . . . .	26
5.4.1.1. PANA AVPs . . . . .	27
5.4.1.2. PANA Phases . . . . .	27
5.4.1.3. PANA session state parameters . . . . .	29
6. Wire-Visible Constraints . . . . .	31
7. Wire-Invisible Constraints . . . . .	31
8. IANA Considerations . . . . .	32
9. Security Considerations . . . . .	32
10. Acknowledgements . . . . .	32
10.1. Contributors . . . . .	32
11. References . . . . .	33
11.1. Normative References . . . . .	33
11.2. Informative References . . . . .	33
Author's Address . . . . .	34

## 1. Introduction

Today's Internet is experienced by users as a set of applications, such as email, instant messaging, and social networks. There are substantial differences in performance between the various end devices with these applications, but in general end devices participating in the Internet today are considered to have relatively high performance.

More and more communications technology is being embedded into our environment. Different types of devices in our buildings, vehicles, equipment and other objects have a need to communicate. It is expected that most of these devices will employ the Internet Protocol suite. The term "Internet of Things" denotes a trend where a large number of devices directly benefit from communication services that use Internet protocols. Many of these devices are not primarily computing devices operated by humans, but exist as components in buildings, vehicles, and the environment. There will be a lot of variation in the computing power, available memory, communications bandwidth, and other capabilities between different types of these devices. With many low-cost, low-power and otherwise constrained devices, it is not always easy to embed all the necessary features.

Historically, there has been a trend to invent special "light-weight" protocols to connect the most constrained devices. However, much of this development can simply run on existing Internet protocols, provided some attention is given to achieving light-weight implementations. In some cases the new, constrained environments can indeed benefit from protocol optimizations and additional protocols that help optimize Internet communications and lower the computational requirements. Examples of IETF standardization efforts targeted for these environments include the "IPv6 over Low power WPAN (6LoWPAN)", "Routing Over Low power and Lossy networks (ROLL)", and "Constrained RESTful Environments (CoRE)" working groups. More generally, however, techniques are required to implement both these optimized protocols as well as the other protocols of the Internet protocol suite in a way that makes them applicable to a wider range of devices.

### 1.1. Objectives

The present document, a product of the IETF Light-Weight Implementation Guidance (LWIG) Working Group, focuses on helping the implementers of the smallest devices. The goal is to be able to build minimal yet interoperable IP-capable devices for the most constrained environments.

Building a small implementation does not have to be hard. Many small devices use stripped down versions of general purpose operating systems and their TCP/IP stacks. However, there are implementations that go even further in minimization and can exist in as few as a couple of kilobytes of code, as on some devices this level of optimization is necessary. Technical and cost considerations may limit the computing power, battery capacity, available memory, or communications bandwidth that can be provided. To overcome these limitations the implementers have to employ the right hardware and software mechanisms. For instance, certain types of memory management or even fixed memory allocation may be required. It is also useful to understand what is necessary from the point of view of the communications protocols and the application employing them. For instance, a device that only acts as a client or only requires one connection can simplify its TCP implementation considerably.

The purpose of this document is to collect experiences from implementers of IP stacks in constrained devices. The focus is on techniques that have been used in actual implementations and do not impact interoperability with other devices. The techniques shall also not affect conformance to the relevant specifications. We describe implementation techniques for reducing complexity, memory footprint, or power usage.

The topics for this working group will be chosen from Internet protocols that are in wide use today, such as IPv4 and IPv6; UDP and TCP; ICMPv4/v6, MLD/IGMP and ND; DNS and DHCPv4/v6; TLS, DTLS and IPsec; as well as from the optimized protocols that result from the work of the 6LoWPAN, RPL, and CoRE working groups. This document will be helpful for the implementers of new devices or for the implementers of new general-purpose small IP stacks. It is also expected that the document will increase our knowledge of what existing small implementations do, and will help in the further optimization of the existing implementations. In areas where the considerations for small implementations have already been documented in an accessible way, we will refer to those documents instead of duplicating the material here.

Generic hardware design advice and software implementation techniques are outside the scope of this document. Protocol implementation experience, however, is the focus. There is no intention to describe any new protocols or protocol behavior modifications beyond what is already allowed by existing RFCs, because it is important to ensure that different types of devices can work together. For example, implementation techniques relating to security mechanisms are within scope, but mere removal of security functionality from a protocol is rarely an acceptable approach.

#### 1.2. Call for contributions

The present draft of the document is an outline that will grow with the contributions received, which are expressly invited. As this document focuses on experience from existing implementations, this requires implementer input; in particular, participation is required from the implementers of existing small IP stacks. "Small" here is intended to be applicable approximately to what is described in Section 2 -\u002D where it is more important that the technique described is grounded in actual experience than that the experience is actually from a (very) constrained system.

Only a few subsections are fleshed out in this initial draft; additional subsections will quickly be integrated from additional contributors.

#### 1.3. Terminology used in this document

The present document has originally also been used to develop pertinent terminology. This has been factored out into a separate document, [I-D.ietf-lwig-terminology], which is now a prerequisite to reading the present document.

#### 1.4. Scope of the present document

Using this terminology, we can now more precisely define the scope of the present document:

This document is about implementation techniques that enable constrained nodes to form constrained node networks.

Delay-Tolerant Networks (DTNs) are out of scope. (See Section 1.1 above for a further list of non-goals.)

#### 1.5. Terminology boilerplate

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. As this is an informational document, the [RFC2119] keywords will only be used to underscore requirements where similar key words apply in the context of the specifications the light-weight implementation of which is being discussed.

The term "byte" is used in its now customary sense as a synonym for "octet".

### 2. Drawing the Landscape

There is not a single kind of constrained, Internet-connected device. To the contrary, the trend is towards much more functional variety of such devices than is customary today in the Internet. The terminology document [I-D.ietf-lwig-terminology] introduces a number of terms that will be used here to locate some of the technique described in the following sections within certain areas of applications.

#### 2.1. Design Objectives

- o Consideration for design or implementation approaches for implementation of IP stacks for constrained devices will be impacted by the RAM usage for these designs. Here the consideration is what is the best approach to minimize overhead.
- o In addition, the impact on throughput in terms of IP protocol implementation must take into consideration the methods that minimize overhead but balance performance requirements for the light-weight constrained devices.

- o Protocol implementation must consider its impact on CPU utilization. Here guidance will be provided on how to minimize tasks that require additional CPU execution time.

How does the implementation of the IP stack effect the application both in terms of performance but also of those same attributes and requirements (RAM, CPU usage, etc.) that we are examining for the IP protocol stack?

From performing a synthesis of implementation experiences we will be able to understand and document the benefits and consequences of varied approaches. Scaling code and selected approaches in terms of scaling from, say, a 8-bit micro to a 16-bit micro. Such scaling for the approach will aid in the development of single code base when possible.

## 2.2. Implementation Styles

Compared to personal computing devices, constrained devices tend to make use of quite different classes of operating systems, if that term is even applicable.

...

- o Single-threaded/giant mainloop
- o Event-driven vs. threaded/blocking
  - \* The usual multi-threaded model blocks a thread on primitives such as connect(), accept() or read() until an external event takes place. This model is often thought to consume too much RAM and CPU processing.
  - \* The event driven model uses a non-blocking approach: E.g., when an application interface sends a message, the routine would return immediately (before the message is sent). A call-back facility notifies the application or calling code when the desired processing is completed. Here the benefit is that no thread context needs to be preserved for long periods of time.
- o Single/multiple processing elements
- o E.g., separate radio/network processor

Introduce these briefly: Some techniques may be applicable only to some of these styles!

### 2.3. Roles of nodes

Constrained nodes are by necessity more specialized than general purpose computing devices; they may have a quite specific role. Some implementation techniques may also

- o Constrained nodes
- o Nodes talking to constrained nodes
- o Gateways/Proxies

In all these cases, constrained nodes that are "sleepy" pose additional considerations. (Explain sleepy...) E.g., a node talking to a sleepy node may need to make special arrangements; this is even more true where a gateway or proxy interfaces the general Internet

- o Bandwidth/latency considerations

### 2.4. Overview over the document

The following sections will first go through a number of specific protocol layers, starting from layers of the data plane (link adaptation, network, transport, application), followed by control plane protocol layers (link layer support, network layer and routing, host configuration and lookup services). We then look at security protocols (general cryptography considerations, transport layer security, network layer security, network access control). Finally, we discuss some specific, cross-layer concerns, some "wire-visible", some of concern within a specific implementation. Clearly, many topics could be discussed in more than one place in this structure. The objective is not to have something for each of the potential topics, but to document the most valuable experience that may be available.

## 3. Data Plane Protocols

### 3.1. Link Adaptation Layer

#### 6LoWPAN

#### 3.1.1. Fragmentation in a 6LoWPAN Route-Over Configuration

Author: Carsten Bormann

6LoWPAN [RFC4944] is an adaptation layer that maps IPv6 with its minimum MTU of 1280 bytes to IEEE 802.15.4, which has a physical layer MTU of only 127 bytes (some of which are taken by MAC layer and

adaptation layer headers). Therefore, the adaptation layer provides a fragmentation and reassembly scheme that can fragment a single IPv6 packet of up to 1280 bytes into multiple adaptation layer fragments of up to 127 bytes each (including MAC and adaptation layer overhead).

In a route-over configuration, implementing this adaptation layer fragmentation scheme straightforwardly means that reassembly and then fragmentation are performed at each forwarding hop. As fragments from several packets may be arriving interleaved with each other, this approach requires buffer space for multiple MTU-size IPv6 packets.

In a mesh-under configuration, adaptation layer fragments can be forwarded independently of each other. It would be preferable if something similar were possible for route-over. Complete independence in forwarding of adaptation layer fragments is not possible for route-over, however, as the layer-3 addresses needed for forwarding are in the initial bytes of the IPv6 header, which is present only in the first fragment of a larger packet.

Instead of performing a full reassembly, implementations may be able to optimize this process by not keeping a full reassembly buffer, but just a runt buffer (called "virtual reassembly buffer" in [WEI]) for each IP packet. This buffer caches only the datagram\_tag field (as usual combined with the sender's link layer address, the destination's link layer address and the datagram\_size field) and the IPv6 header including the relevant addresses. Initial fragments are then forwarded independently (after header decompression/compression) and create a runt reassembly buffer. Non-initial fragments (which don't require header decompression/compression in 6LoWPAN) are matched against the runt buffers by datagram\_tag etc. and forwarded if an IPv6 address is available. (This simple scheme may be complicated a bit if header decompression/compression of the initial fragment causes an overflow of the physical MTU; in this case some overflow data may need to be stored in the runt buffers to be combined with further fragments or may simply be forwarded as a separate additional fragment.)

If non-initial fragments arrive out of order before the initial fragment, a route-over router may want to keep the contents of the non-initial fragments until the initial fragment is available, which does need some buffer space. If that is not available, a more constrained route-over router may simply discard out-of order non-initial fragments, possibly taking note that there is no point in forwarding any more fragments with the same combination of 6LoWPAN datagram\_tag field, L2 addresses and datagram\_size.

Runt buffers should time out like full reassembly buffers, and may either keep a map of fragments forwarded or they may simply be removed upon forwarding the final fragment, assuming that no out-of-order fragments will follow.

#### 3.1.1.1. Implementation Considerations for Not-So-Constrained Nodes

[RFC4944] makes no explicit mandates about the order in which fragments should be sent. Because it is heavily favored by the above implementation techniques, it is highly advisable for all implementations to always send adaptation layer fragments in natural order, i.e., starting with the initial fragment, continuing with increasing datagram\_offset.

### 3.2. Network Layer

IPv4 and IPv6

### 3.3. Transport Layer

TCP and UDP

Both TCP and UDP employ 16-bit one's-complement checksums to protect against transmission errors. A number of RFCs discuss efficient implementation techniques for computing and updating Internet Checksums [RFC1071] [RFC1141] [RFC1624]. (Updating the Internet Checksum, as opposed to computing it from scratch, may be of interest where a pre-computed packet is provided, e.g., in Flash ROM, and a copy is made in RAM and updated with some current values, or when the actual transmitted packet is composed from pre-defined parts in ROM and new parts in RAM.)

#### 3.3.1. TCP

Ed. Note:

The following outline of a section is an attempt to provide substructure for a future discussion of TCP-related issues based on the TCP Roadmap, [RFC4614]. The indented text, as well as the RFC citations, are copied (and redacted) from there; this certainly needs to be refined in a future version. (Some additional adaptation of the material may also be required as RFC 2581 was since obsoleted by RFC 5681, and RFC 3782 was obsoleted by RFC 6582.)

Author: Yuanchen Ma

In [RFC4614], the TCP related RFCs are summarized. Some RFCs describe absolutely required TCP behaviors for proper functioning and

interoperability. Further RFCs describe strongly encouraged, but non-essential, behaviors. There are also experimental extensions that are not yet standard practices, but that potentially could be in the future.

In this subsection, the influence of resource constrained nodes on TCP implementations are summarized according to the lists of [RFC4614].

#### 3.3.1.1. Absolutely required TCP behaviors for proper functioning and interoperability

RFC 793 S: "Transmission Control Protocol", STD 7 (September 1981)

In RFC793, the TCP state machine and event processing, and TCP's semantics for data transmission, reliability, flow control, multiplexing, and acknowledgment. For this part, the constraint of memory will limit the multiplexing capability of TCP. /\_text needed for RFC793\_/

RFC 1122 S: "Requirements for Internet Hosts - Communication Layers" (October 1989)

RFC 2460 S: "Internet Protocol, Version 6 (IPv6) Specification" (December 1998)

RFC 2873 S: "TCP Processing of the IPv4 Precedence Field" (June 2000)

This document [RFC2873] removes from the TCP specification all processing of the precedence bits of the TOS byte of the IP header.

These three RFCs mandate the support for IPv6 and TOS in IP header, which are a must for resource constrained node to implement.

RFC 2581 S: "TCP Congestion Control" (April 1999)

Although RFC 793 did not contain any congestion control mechanisms, today congestion control is a required component of TCP implementations. This document [RFC2581] defines the current versions of Van Jacobson's congestion avoidance and control mechanisms for TCP, based on his 1988 SIGCOMM paper [Jac88]. RFC 2001 was a conceptual precursor that was obsoleted by RFC 2581.

A number of behaviors that together constitute what the community refers to as "Reno TCP" are described in RFC 2581.

RFC 1122 mandates the implementation of a congestion control mechanism, and RFC 2581 details the currently accepted mechanism. RFC 2581 differs slightly from the other documents listed in this section, as it does not affect the ability of two TCP endpoints to communicate; however, congestion control remains a critical component of any widely deployed TCP implementation and is required for the avoidance of congestion collapse and to ensure fairness among competing flows.

RFC 2988 S: "Computing TCP's Retransmission Timer" (November 2000)

Abstract: "This document defines the standard algorithm that Transmission Control Protocol (TCP) senders are required to use to compute and manage their retransmission timer.

### 3.3.1.2. Strongly encouraged, but non-essential, behaviors of TCP

RFC 1323 S: "TCP Extensions for High Performance" (May 1992)

This document [RFC1323] defines TCP extensions for window scaling, timestamps, and protection against wrapped sequence numbers, for efficient and safe operation over paths with large bandwidth-delay products.

RFC 2675 S: "IPv6 Jumbograms" (August 1999)

IPv6 supports longer datagrams than were allowed in IPv4.

RFC 3168 S: "The Addition of Explicit Congestion Notification (ECN) to IP" (September 2001)

#### 3.3.1.2.1. Congestion Control and Loss Recovery Extensions

RFC 3042 S: "Enhancing TCP's Loss Recovery Using Limited Transmit" (January 2001)

Abstract: "This document proposes Limited Transmit, a new Transmission Control Protocol (TCP) mechanism that can be used to more effectively recover lost segments when a connection's congestion window is small

RFC 3390 S: "Increasing TCP's Initial Window" (October 2002)

This document [RFC3390] updates RFC 2581 to permit an initial TCP window of three or four segments during the slow-start phase, depending on the segment size.

RFC 3782 S: "The NewReno Modification to TCP's Fast Recovery Algorithm" (April 2004)

This document [RFC3782] specifies a modification to the standard Reno fast recovery algorithm, whereby a TCP sender can use partial acknowledgments to make inferences determining the next segment to send in situations where SACK would be helpful but isn't available.

#### 3.3.1.2.2. SACK-Based Loss Recovery and Congestion Control

RFC 2018 S: "TCP Selective Acknowledgment Options" (October 1996)

This document [RFC2018] defines the basic selective acknowledgment (SACK) mechanism for TCP.

RFC 2883 S: "An Extension to the Selective Acknowledgement (SACK) Option for TCP" (July 2000)

This document [RFC2883] extends RFC 2018 to cover the case of acknowledging duplicate segments.

RFC 3517 S: "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP" (April 2003)

#### 3.3.1.2.3. Dealing with Forged Segments

RFC 1948 I: "Defending Against Sequence Number Attacks" (May 1996)

RFC 2385 S: "Protection of BGP Sessions via the TCP MD5 Signature Option" (August 1998)

#### 3.3.1.3. Experimental extensions that are not yet standard practices

The experimental extensions are not mature yet. The contents need to be validated to be safe and logical behavior. It is not recommended for the resource constrained node to implement.

#### 3.3.1.4. Others

RFC 2923 I: "TCP Problems with Path MTU Discovery" (September 2000)

From abstract: "This memo catalogs several known Transmission Control Protocol (TCP) implementation problems dealing with Path Maximum Transmission Unit Discovery (PMTUD), including the long-standing black hole problem, stretch acknowledgements (ACKs) due to confusion between Maximum Segment Size (MSS) and segment size, and MSS advertisement based on PMTU." [RFC2923]

### 3.4. Application Layer

#### 3.4.1. General considerations about Application Programming Interfaces (APIs)

Author: Carl Williams

Constrained devices are not necessarily in a position to use APIs that would be considered "standard" for less constrained environments (e.g., Berkeley sockets or those defined by POSIX).

When an API implements a protocol, this can be based on proxy methods for remote invocations that underneath rely on the communication protocol. One of the roles of the API can be exactly to hide the detail of the transport protocol.

Changes to the lower layers will be made to implement light-weight stacks so this impacts that implementation and inter-workings with the API. Similar considerations such as RAM, CPU utilization and performance requirements apply to the API and its use of the lower layer resources (i.e., buffers).

Considerations for the proper approach for a developer to request services from an application program need to be explored and documented. Such considerations will allow the progression of a common consistent networking paradigm without inventing a new way of programming these devices.

In addition, such considerations will take into account the inter-working of the API with the protocols. Protocols are more complex to use as they are less direct and take a lot of serializing, de-serializing and dispatching type logic.

So the connection of the API and the protocols on a constrained device becomes even more important to balance the requirements of RAM, CPU and performance.

`_** Here we will proceed to collect and document ... insert experiences from existing API on constrained devices (TBD) **_`

#### 3.4.2. Constrained Application Protocol (CoAP)

Author: Olaf Bergmann

The Constrained Application Protocol [I-D.ietf-core-coap] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios therefore include building automation and the Internet of Things.

The major design objectives have been set on small protocol overhead, robustness against packet loss, and 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 architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [RFC2616].

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 inevitable for the core protocol and thus must be implemented on every node has been kept very small to avoid unnecessary accumulation of optional features. Options that -\u002D when present -\u002D 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 trade-off between robustness and latency of constrained nodes on one hand and resource demands (such as battery consumption, dynamic memory needs and static code-size) on the other. This section gives some guidance on possible strategies to solve this trade-off for very constrained nodes (Class 1 in [I-D.ietf-lwig-terminology]). The main focus is on servers as this is deemed the predominant case where CoAP applications are faced with tight resource constraints.

Additional considerations for the implementation of CoAP on tiny sensors are given in [I-D.arkko-core-sleepy-sensors].

#### 3.4.2.1. Message Layer Processing

For constrained nodes of Class 1 or even Class 2, limiting factors for (wireless) network communication usually are RAM size and battery lifetime. Most applications therefore try to avoid dealing with fragmented packets on the network layer and minimize internal buffer space for both transmit and receive operations. One of the most expensive operations hence is the retransmission of messages as it implies additional energy consumption for the (radio) network interface and occupied RAM storage for the send buffer.

Where multi-threading is not an option at all because no full-fledged operating system is present, all operations are triggered by a big

main loop in a send-receive-dispatch cycle. To implement the packet retransmission, CoAP implementations at least need a separate send buffer and a decent notion of time, e.g. as a strictly monotonic increasing tick counter. 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.

A similar strategy holds for confirmable messages with separate responses. This concept entitles CoAP servers to return an empty acknowledgement to indicate that a confirmable request has been understood and is being processed. Once a proper response has been generated to fulfill the request, it is sent back as a confirmable message as well. The server implementation in this case must be able to map retransmissions of the original request to the ongoing operation and provide the client-selected Token to map between original request and the separate response.

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.

#### 3.4.2.2. Message Parsing

Both CoAP clients and servers must construct outgoing CoAP PDUs and parse incoming messages. The basic message header consists of only four octets and thus can be mapped easily to an internal data structure, considering the actual byte order of the host. 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. The delta-encoded option number indicates the number of left-shift operations to apply on a bit mask to set the corresponding bit.

In addition, the byte index of every option is added to a sparse list (e.g. a one-dimensional array) for fast retrieval. This particularly enables efficient reduced-function 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 option 9 (Uri-Path) will be overwritten to hold a pointer to the last occurrence of that option, i.e., only the last path component actually matters. (Of course, this requires choosing resource names where the combination of (final Uri-Path component, final Uri-Query component) is server-wide unique.

Note: Where skipping all but the last path segment is not feasible for some reason, resource identification could be ensured by some hash value calculated over the path segments. For each segment encountered, the stored hash value is updated by the current option value. This works if a cheap perfect hashing scheme can be found for the resource names.

Once the option list has been processed at least up to the highest option number that is supported by the application, any known critical option and all elective options can be masked out 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 the remaining options also must be processed to add further critical options included in the original request.)

#### 3.4.2.3. Storing Used Message IDs

If CoAP is used directly on top of UDP (i.e., in NoSec mode), it needs to cope with the fact that the UDP datagram transport can reorder and duplicate messages. (In contrast to UDP, DTLS has its own duplicate detection.) CoAP has been designed with protocol functionality such that rejection of duplicate messages is always possible. It is at the discretion of the receiver if it actually wants to make use of this functionality. Processing of duplicate messages comes at a cost, but so does the management of the state associated with duplicate rejection. Hence, a receiver may have good reasons to decide not to do the duplicate rejection. 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.

Author: Esko Dijk

CoAP's duplicate rejection functionality can be straightforwardly implemented in a CoAP end-point by storing, for each remote CoAP end-point ("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_` ([I-D.ietf-core-coap] section "Reliable Messages") and the time packets can stay alive in the network.

Such a straightforward implementation is suitable in case other CoAP end-points 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 [I-D.ietf-core-coap] 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 end-point 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 end-points, 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 end-points it communicates with, which partly breaks sequentiality from the receiving CoAP end-point'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	1111011110111	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 end-point 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 end-point 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 end-point uses sequential MIDs and in response improve efficiency by switching its mode to the bitfield based storage.

### 3.4.3. (Other Application Protocols...)

## 4. Control Plane Protocols

### 4.1. Link Layer Support

ARP, ND; 6LoWPAN-ND

### 4.2. Network Layer

ICMP, ICMPv6, IGMP/MLD

#### 4.3. Routing

RPL, AODV/DYMO, OLSRv2

#### 4.4. Host Configuration and Lookup Services

DNS, DHCPv4, DHCPv6

#### 4.5. Network Management

SNMP, netconf?

##### 4.5.1. SNMP

Author: Brinda M C

This section describes an approach for developing a light-weight SNMP agent for resource constrained devices running the 6LoWPAN/RPL protocol stack. The motivation for the work is driven by two major factors:

- o SNMP plays a vital role in monitoring and managing any operational network; 6LoWPAN based WSN is no exception to this.
- o There is a need for building a light-weight SNMP agent which consumes less memory and less computational resources.

The following subsections are organized as follows:

- o Section 4.5.1.1 provides some background.
- o In Section 4.5.1.2, we revisit existing SNMP implementation in the context of memory constrained devices.
- o In Section 4.5.1.3, we present our approach for building a memory efficient SNMP agent.
- o Using a realistic example, in Section 4.5.1.4, we illustrate how the proposed method can be implemented.
- o In Section 4.5.1.5, we explore a few ideas which can further help in improving the memory utilization.

##### 4.5.1.1. Background

Our initial SNMP agent implementation was completely based on Net-SNMP, well-known open-source network monitoring and management software. After porting the agent on to the TelosB mote, we observed that it occupies a text program memory of more than 8 KiB on TinyOS and Contiki OS platforms. (Note that both these platforms already use compiler optimizations to minimize the memory footprint.) 8 KiB is already non-negligible given the 48 KiB program memory limit of TelosB. Added to this, the memory taken up by 6LoWPAN and the related protocol stacks are ever growing, causing serious memory crunch in the resource constrained devices. We reached a situation where we could not build an image on the TinyOS/Contiki OS platforms with our SNMP agent.

We came across SNMPv1 agent implementations elsewhere in the literature which also report similar memory consumption. This motivated us to have a re-look at the existing SNMP agent implementation, and explore the possibility of an alternate implementation using altogether a different approach.

#### 4.5.1.2. Revisiting SNMP implementation for resource constrained devices

If we look at a typical SNMP agent implementation, we can see that much of the memory consuming code is pertaining to ASN.1 related SNMP PDU parsing and SNMP PDU build operations. The SNMP parsing mainly recovers various fields from the incoming PDU, such as the OIDs, whereas the SNMP PDU build is the reverse operation of building the response PDU from the OIDs.

The key observation is that, for a given MIB definition, an OID of interest contained in the incoming SNMP PDU is already available, albeit in an encoded form. This enables identifying the OID from the packet in its "raw" form, simplifying parser operation.

We also can make use of this observation while building the response SNMP PDU. For a given MIB definition, we can think of statically having a pre-composed ASN.1 encoded version of OIDs, and use them while constructing the response SNMP PDU.

#### 4.5.1.3. Proposed approach for building an memory efficient SNMP agent

As noted in the previous section, since an SNMP OID is already contained in the incoming network PDU, we came up with a simple OID signature identification method performed directly on the network PDU through simple memory comparisons and table look-ups. Once the OID has been identified from the packet "in situ", the corresponding per-OID processing is carried out. Through this scheme we completely eliminated expensive SNMP parse operations.

For the SNMP PDU build, we use `_pre-encoded_` OID variables which can simply be plugged into the network SNMP response packet directly depending on the request OID. Now that the expensive build operation is taken care, what remains is the construction of the overall SNMP pdu which can be built through simple logic. Through this scheme we completely eliminated expensive SNMP build operations.

Based on these ideas, we have re-architected our original SNMP agent implementation and with our new implementation we were able to bring down its text memory usage all the way down to 4 KiB from the native SNMP agent implementation which occupied 8 KiB.

#### 4.5.1.3.1. Discussion on memory usage

With respect to the memory usage, while we have achieved major reduction in terms of text program memory, which occupies a major chunk of memory, a question might come to mind with regard to the static memory allocation for maintaining the tables. We found that this is not very significant to start with. Through an efficient table representation, we further optimized the memory consumption. We could do so because a typical OID description is mainly dominated by a fixed part of the hierarchy. This enables us to define few static prefixes, each corresponding to a particular hierarchy level of the OID. In the context of 6LoWPAN, it can be expected that the number of hierarchy levels will be small.

#### 4.5.1.4. Example

This section illustrates the simplicity and practicality of our approach with an example. Let us consider the fragment of a representative MIB definition depicted in Figure 1

```
iso
|
org
|
dod
|
internet
|
mgmt.mib-2
|
lowpanMIB
|
+--lowpanPrimaryStatistics(10)
|
+--PrimeStatsEntry(1)
|
```

```

+--- -R-- INTEGER    lowpanMoteBatteryVoltageP(1)
+--- -R-- Counter    lowpanFramesReceivedP(2)
+--- -R-- Counter    lowpanFramesSentP(3)
+--- -R-- Counter    ipv6ForwardedMsgP(4)
+--- -R-- Counter    OUTSolicitationP(5)
+--- -R-- Counter    OUTAdvertisementP(6)

```

Figure 1: A fragment of a MIB hierarchy

#### 4.5.1.4.1. Optimized SNMP Parsing

Let us consider a GET request for the OIDs `lowpanMoteBatteryVoltageP` and `lowpanFramesSentP`. Corresponding to these OIDs, a C array dump of the network PDU of SNMP packet with two OIDs in a variable binding would look as in Figure 2.

```

char snmp_get_req_pkt[] = {
    0x30, 0x81, 0x3d, 0x02, 0x01, 0x00, 0x04, 0x06,
    0x70, 0x75, 0x62, 0x6c, 0x69, 0x63, 0xa0, 0x30,
    0x02, 0x04, 0x28, 0x29, 0xe4, 0x5d, 0x02, 0x01,
    0x00, 0x02, 0x01, 0x00, 0x30, 0x22, 0x30, 0x0f,
    0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02, 0x01, 0x83,
    0x90, 0x12, 0x0a, 0x01, 0x01, 0x05, 0x00, 0x30,
    0x0f, 0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02, 0x01,
    0x83, 0x90, 0x12, 0x0a, 0x01, 0x03, 0x05, 0x00 };

```

Figure 2: An SNMP packet, represented in C

Inspecting the above packet, we see that the main components of the PDU are:

1. Version (SNMPv1): [0x02, 0x01, 0x00]
2. Community Name ("public"): [0x04, 0x06, 0x70, 0x75, 0x62, 0x6c, 0x69, 0x63]
3. ASN.1 encoded OIDs for `lowpanMoteBatteryVoltageP`, and `lowpanFramesReceivedP`:
  - \* [0x30, 0x0f, 0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02, 0x01, 0x83, 0x90, 0x12, 0x0a, 0x01, 0x01, 0x05, 0x00]
  - \* [0x30, 0x0f, 0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02, 0x01, 0x83, 0x90, 0x12, 0x0a, 0x01, 0x03, 0x05, 0x00]

There is a significant overlap between the two OIDs, which can be used to simplify the parsing process. We can, for instance, define one statically initialized array containing elements common between

these OIDs. Using this notion of common prefix idea, we can come up with an optimized table and the OID identification then boils down to simple memory comparisons within this table. The optimized table construction will also result in scalability.

#### 4.5.1.4.2. Optimized SNMP Build

Extending the same approach as described above, we can build the GET response by plugging in pre-encoded OIDs into the response packets. So, corresponding to the GET request for the OIDs as given in section 4.1, we can define C arrays containing pre-encoded OIDs which can go into the response packet as in Figure 3.

```
pdu_batt_volt[] = {
    0x30, 0x11, 0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02,
    0x01, 0x83, 0x90, 0x12, 0x0a, 0x01, 0x01, 0x02,
    0x02, 0x00, 0x00 };

pdu_frames_sent[] = {
    0x30, 0x11, 0x06, 0x0b, 0x2b, 0x06, 0x01, 0x02,
    0x01, 0x83, 0x90, 0x12, 0x0a, 0x01, 0x03, 0x41,
    0x02, 0x00, 0x00 };
```

Figure 3: Pre-encoded OIDs

Since the ASN.1 basic encoding rules are in TLV format, the offset within the encoded OID where the value needs to be filled-in can be obtained from the length field.

The table size optimization discussed in the previous section can be applied here, too.

Note: Though we have taken a simple example to illustrate the efficacy of the proposed approach, the ideas presented here can easily be extended to other scenarios as well.

#### 4.5.1.5. Further improvements

A few simple methods can reduce the code size as well as generate computationally inexpensive code. These methods might sound obvious and trivial but are important for constrained devices.

- o If possible, avoid using memory consuming data types such as floating point while representing a monitored variable when an equivalent representation of the same that occupies less memory is adequate. For example, while a battery voltage indication could take a fractional value between 0 and 3 V, opt for an 8-bit quantized value.

- o Using meta data in the MIB definition instead of absolute numbers can bring down the memory and processing significantly and can improve scalability too especially for a large scale WSN deployments. Using the same example of battery voltage, one might think of an OID which represents fewer levels of the battery voltage signifying high, medium, low, very low.
- o While a multi-level hierarchy for MIB definition might improve OID segregation the flip side is that it increases the overall length of the OID and results in extra memory and processing overhead. One may have to make a judicious choice while coming up with the MIB.

#### 4.5.1.6. Conclusion

This subsection proposes a simple SNMP packet processing based approach for building a light-weight SNMP agent. While there is scope for further improvement, we believe that the proposed method can be a reasonably good starting point for resource constrained 6LoWPAN based networks.

### 5. Security protocols

#### 5.1. Cryptography for Constrained Devices

#### 5.2. Transport Layer Security

TLS, DTLS, ciphersuites, certificates

#### 5.3. Network Layer Security

IPsec, IKEv2, transforms

Advice for a minimal implementation of IKEv2 can be found in [I-D.kivinen-ipsecme-ikev2-minimal].

#### 5.4. Network Access Control

(PANA, EAP, EAP methods)

##### 5.4.1. PANA

Author: Mitsuru Kanda

PANA [RFC5191] provides network access authentication between clients and access networks. The PANA protocol runs between a PANA Client (PaC) and a PANA Authentication Agent (PAA). PANA carries UDP encapsulated EAP [RFC3748] and includes various operational options.

From the point of view of minimal implementation, some of these are not necessary for constrained devices. This section describes a minimal PANA implementation for these devices.

The minimization objective for this implementation mainly targets PaCs because constrained devices often are installed as network clients, such as sensors, metering devices, etc.

#### 5.4.1.1. PANA AVPs

Each PANA message can carry zero or more AVPs (Attribute-Value Pairs) within its payload. [RFC5191] specifies nine types of AVPs (AUTH, EAP-Payload, Integrity-Algorithm, Key-Id, Nonce, PRF-Algorithm, Result-Code, Session-Lifetime, and Termination-Cause). All of them are required by all minimal implementations. But there are some notes.

Integrity-Algorithm AVP and PRF-Algorithm AVP:

All PANA implementations MUST support AUTH\_HMAC\_SHA1\_160 for PANA message integrity protection and PRF\_HMAC\_SHA1 for pseudo-random function (PRF) specified in [RFC5191]. Both of these are based on SHA-1, which therefore needs to be implemented in a minimal implementation.

Nonce AVP:

As the basic hash function is SHA-1, including a nonce of 20 bytes in the Nonce AVP is appropriate ([RFC5191], section 8.5).

#### 5.4.1.2. PANA Phases

A PANA session consists of four phases -\u002D Authentication and authorization phase, Access phase, Re-Authentication phase, and Termination phase.

Authentication and authorization phase:

There are two types of PANA session initiation, PaC-initiated session and PAA-initiated session. The minimal implementation must support PaC-initiated session and does not need to support PAA-initiated session. Because a PaC (a constrained device) which may be a sleeping device, can not receive an unsolicited PANA-Auth-Request message from a PAA (PAA-initiated session).

EAP messages can be carried in PANA-Auth-Request and PANA-Auth-Answer messages. In order to reduce the number of messages, "Piggybacking EAP" is useful. Both the PaC and PAA should include EAP-Payload AVP

in each of PANA-Auth-Request and PANA-Auth-Answer messages as much as possible. Figure 4 shows an example "Piggybacking EAP" sequence of the Authentication and authorization phase.

```

PaC      PAA  Message(sequence number)[AVPs]
-----
----->    PANA-Client-Initiation(0)
<-----    PANA-Auth-Request(x)[PRF-Algorithm,Integrity-Algorithm]
              // The 'S' (Start) bit set
----->    PANA-Auth-Answer(x)[PRF-Algorithm, Integrity-Algorithm]
              // The 'S' (Start) bit set
<-----    PANA-Auth-Request(x+1)[Nonce, EAP-Payload]
----->    PANA-Auth-Answer(x+1)[Nonce, EAP-Payload]
<-----    PANA-Auth-Request(x+2)[EAP-Payload]
----->    PANA-Auth-Answer(x+2)[EAP-Payload]
<-----    PANA-Auth-Request(x+3)[Result-Code, EAP-Payload,
              Key-Id, Session-Lifetime, AUTH]
              // The 'C' (Complete) bit set
----->    PANA-Auth-Answer(x+3)[Key-Id, AUTH]
              // The 'C' (Complete) bit set

```

Figure 4: Example sequence of the Authentication and authorization phase for a PaC-initiated session (using "Piggybacking EAP")

Note: It is possible to include an EAP-Payload in both the PANA-Auth-Request and PANA-Auth-Answer messages with the 'S' bit set. But the PAA should not include an EAP-Payload in the PANA-Auth-Request message with the 'S' bit set in order to stay stateless in response to a PANA-Client-Initiation message.

#### Access phase:

After Authentication and authorization phase completion, the PaC and PAA share a PANA Security Association (SA) and move Access phase. During Access phase, [RFC5191] describes both the PaC and PAA can send a PANA-Notification-Request message with the 'P' (Ping) bit set for the peer's PANA session liveness check (a.k.a "PANA ping"). From the minimal implementation point of view, the PAA should not send a PANA-Notification-Request message with the 'P' (Ping) bit set to initiate PANA ping since the PaC may be sleeping. The PaC does not need to send a PANA-Notification-Request message with the 'P' (Ping) bit set for PANA ping to the PAA periodically and may omit the PANA ping feature itself if the PaC can detect the PANA session failure by other methods, for example, network communication failure. In conclusion, the PaC does not need to implement the periodic liveness check feature sending PANA ping but a PaC that is awake should respond to a incoming PANA-Notification-Request message with the 'P' (Ping) bit set for PANA ping as possible.

## Re-Authentication phase:

Before PANA session lifetime expiration, the PaC and PAA MUST re-negotiate to keep the PANA session. This means that the PaC and PAA enter Re-Authentication phase. Also in the Authentication and authorization phase, there are two types of re-authentication. The minimal implementation must support PaC-initiated re-authentication and does not need to support PAA-initiated re-authentication (again because the PaC may be a sleeping device). "Piggybacking EAP" is also useful here and should be used as well. Figure 5 shows an example "Piggybacking EAP" sequence of the Re-Authentication phase.

PaC	PAA	Message(sequence number)[AVPs]
----->		PANA-Notification-Request(q)[AUTH] // The 'A' (re-Authentication) bit set
<-----		PANA-Notification-Answer(q)[AUTH] // The 'A' (re-Authentication) bit set
<-----		PANA-Auth-Request(p)[EAP-Payload, Nonce, AUTH]
----->		PANA-Auth-Answer(p)[EAP-Payload, Nonce, AUTH]
<-----		PANA-Auth-Request(p+1)[EAP-Payload, AUTH]
----->		PANA-Auth-Answer(p+1)[EAP-Payload, AUTH]
<-----		PANA-Auth-Request(p+2)[Result-Code, EAP-Payload, Key-Id, Session-Lifetime, AUTH] // The 'C' (Complete) bit set
----->		PANA-Auth-Answer(p+2)[Key-Id, AUTH] // The 'C' (Complete) bit set

Figure 5: Example sequence of the Re-Authentication phase for a PaC-initiated session (using "Piggybacking EAP")

## Termination Phase:

The PaC and PAA should not send a PANA-Termination-Request message except for explicitly terminating a PANA session within the lifetime. Both the PaC and PAA know their own PANA session lifetime expiration. This means the PaC and PAA should not send a PANA-Termination-Request message when the PANA session lifetime expired because of reducing message processing cost.

## 5.4.1.3. PANA session state parameters

All PANA implementations internally keep PANA session state information for each peer. At least, all minimal implementations need to keep PANA session state parameters below (in the second column storage sizes are given in bytes):

```
+-----+-----+-----+
```

State Parameter	Size	Comment
PANA Phase Information	1	Used for recording the current PANA phase.
PANA Session Identifier	4	
PaC's IP address and UDP port number	6 or 18	IP Address length (4 bytes for IPv4 and 16 bytes for IPv6) plus 2 bytes for UDP port number.
PAA's IP address and UDP port number	6 or 18	IP Address length (4 bytes for IPv4 and 16 bytes for IPv6) plus 2 bytes for UDP port number.
Outgoing message sequence number	4	Next outgoing request message sequence number.
Incoming message sequence number	4	Next expected incoming request message sequence number.
A copy of the last sent message payload	variable	Necessary to be able to retransmit the message (unless it can be reconstructed on the fly).
Retransmission interval	4	
PANA Session lifetime	4	
PaC nonce	20	Generated by PaC and carried in the Nonce AVP.
PAA nonce	20	Generated by PAA and carried in the Nonce AVP.
EAP MSK Identifier	4	
EAP MSK value	*)	Generated by EAP method and used for generating PANA_AUTH_KEY.

PANA_AUTH_KEY	20	Necessary for PANA message protection.
PANA PRF algorithm number	4	Used for generating PANA_AUTH_KEY.
PANA Integrity algorithm number	4	Necessary for PANA message protection.

\*) (Storage size depends on the key derivation algorithm.)

Note: EAP parameters except for MSK have not been listed here. These EAP parameters are not used by PANA and depend on what EAP method you choose.

#### 6. Wire-Visible Constraints

- o Checksum
- o MTU
- o Fragmentation and reassembly
- o Options -\u002D implications of leaving some out
- o Simplified TCP optimized for LLNs
- o Out-of-order packets

#### 7. Wire-Invisible Constraints

- o Buffering
- o Memory management
- o Timers
- o Energy efficiency
- o API
- o Data structures
- o Table sizes (somewhat wire-visible)
- o Improved error handling due to resource overconsumption

## 8. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

## 9. Security Considerations

(TBD.)

## 10. Acknowledgements

Much of the text of the introduction is taken from the charter of the LWIG working group and the invitation to the IAB workshop on Interconnecting Smart Objects with the Internet. Thanks to the numerous contributors. Angelo Castellani provided comments that led to improved text.

### 10.1. Contributors

The RFC guidelines no longer allow RFCs to be published with a large number of authors. As there are many authors that have contributed to the sections of this document, their names are listed in the individual section headings as well as alphabetically listed with their affiliations below.

Name	Affiliation	Contact
Brinda M C	Indian Institute of Science	brinda@ece.iisc.ernet.in
Carl Williams	MCSR Labs	carlw@mcsr-labs.org
Carsten Bormann	Universitaet Bremen TZI	cabo@tzi.org
Esko Dijk	Philips Research	esko.dijk@philips.com
Mitsuru Kanda	Toshiba	mitsuru.kanda@toshiba.co.jp
Olaf Bergmann	Universitaet Bremen TZI	bergmann@tzi.org
Yuanchen Ma	Hitachi (China) R&D Corporation	ycma@hitachi.cn

| ... | ... |  
+-----+-----+-----+-----+

## 11. References

### 11.1. Normative References

- [I-D.ietf-lwig-terminology]  
Bormann, C. and M. Ersue, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-00 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

### 11.2. Informative References

- [I-D.arkko-core-sleepy-sensors]  
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.
- [I-D.ietf-6lowpan-btle]  
Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "Transmission of IPv6 Packets over BLUETOOTH Low Energy", draft-ietf-6lowpan-btle-12 (work in progress), February 2013.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.kivinen-ipsecme-ikev2-minimal]  
Kivinen, T., "Minimal IKEv2", draft-kivinen-ipsecme-ikev2-minimal-01 (work in progress), October 2012.
- [I-D.mariager-6lowpan-v6over-dect-ule]  
Mariager, P. and J. Petersen, "Transmission of IPv6 Packets over DECT Ultra Low Energy", draft-mariager-6lowpan-v6over-dect-ule-02 (work in progress), May 2012.

- [RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.
- [RFC1141] Mallory, T. and A. Kullberg, "Incremental updating of the Internet checksum", RFC 1141, January 1990.
- [RFC1624] Rijssinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4614] Duke, M., Braden, R., Eddy, W., and E. Blanton, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 4614, September 2006.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.
- [WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", ISBN 9780470747995, 2009.

Author's Address

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

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