

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 15, 2014

Z. Cao
China Mobile
X. He
Hitachi (China) Research and Development Corpor
M. Kovatsch
ETH Zurich
H. Tian
China Academy of Telecommunication Research
July 14, 2013

Energy Efficient Implementation of IETF Protocols on Constrained Devices
draft-hex-lwig-energy-efficient-01

Abstract

This document summarizes the problems and current practices of energy efficient protocol implementation on constrained devices, mostly about how to make the protocols within IETF scope behave energy friendly. This document also summarizes the impact of link layer protocol power saving behaviors to the upper layer protocols, so that they can coordinately make the system energy efficient.

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 15, 2014.

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	2
1.1. Conventions used in this document	3
1.2. Terminology	3
2. Overview	3
3. MAC and Radio Duty Cycling	4
3.1. Power Save Services Provided by IEEE 802.11v	5
4. IP Adaption and Transport Layer	6
5. Routing Protocols	7
6. Application Layer	7
7. Cross Layer Optimization	7
8. Summary	8
9. IANA Considerations	8
10. Security Considerations	8
11. References	8
11.1. Normative References	8
11.2. Informative References	9
Authors' Addresses	10

1. Introduction

In many scenarios of embedded systems, the networked system is composed of many battery-powered devices. For example, in an environmental monitoring system or a temperature and humidity monitoring system in the data center, there are no always-on and handy sustained power supplies for the large number of small devices. In such deployment environments, it is necessary to optimize the energy consumption of the entire system, including computing, application layer behavior, and lower layer communication.

Various research efforts have been spent on this "energy efficiency" problem. Most of this research has focused on how to optimize the system's power consumption regarding a certain deployment scenario or how could an existing network function such as routing or security be more energy-efficient. Only few efforts were spent on energy-efficient designs for IETF protocols and standardized network stacks for such constrained devices [I-D.kovatsch-lwig-class1-coap].

The IETF has developed a suite of Internet protocols suitable for such small devices, including 6LoWPAN [RFC6282], 6LoWPAN-ND

[RFC6775], RPL[RFC6550], and CoAP[I-D.ietf-core-coap]. This document tries to summarize the design considerations of making the IETF protocol suite as energy-efficient as possible. While this document does not provide detailed and systematic solutions to the energy efficiency problem, it summarizes the design efforts and analyzes the design space of this problem.

After reviewing the energy-efficient design of each layer, an overall conclusion is summarized. Though the lower layer communication optimization is the key part of energy efficient design, the protocol design at the network and application layers is also important to make the device battery-friendly.

1.1. Conventions used in this document

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

1.2. Terminology

The terminologies used in this document can be referred to [I-D.ietf-lwig-terminology].

2. Overview

The IETF has developed multiple protocols to enable end-to-end IP communication between constrained nodes and fully capable nodes. This work has witnessed the evolution of the traditional Internet protocol stack to a light-weight Internet protocol stack. As show in Figure 1 below, the IETF has developed CoAP as the application layer and 6LoWPAN as the adaption layer to run IPv6 over IEEE 802.15.4 and Bluetooth Low-Energy, with the support of routing by RPL and efficient neighbor discovery by 6LoWPAN-ND.

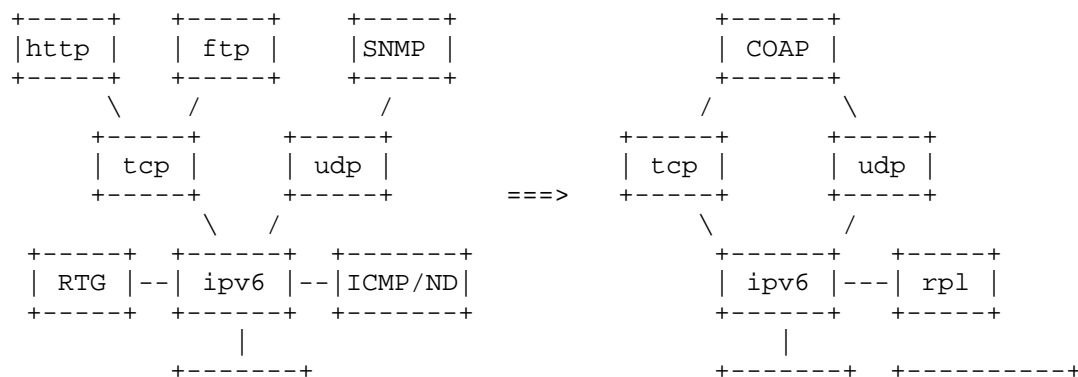




Figure 1: Traditional and Lightweight Internet Protocol Stack

There are comprehensive measurements of wireless communication [Powertrace]. Below we list the energy consumption profile of the most common atom operations on a prevalent sensor node platform. The measurement was based on the Tmote Sky with ContikiMAC as the radio duty cycling algorithm. From the measurement, we can see that optimized transmissions and reception consume almost the same amount of energy. For IEEE 802.15.4 and UWB radios, transmitting is actually even cheaper than receiving. Only for broadcast and non-synchronized communication transmissions become costly in terms of energy because they need to flood the medium for a long time.

Activity	Energy (uJ)
Broadcast reception	178
Unicast reception	222
Broadcast transmission	1790
Non-synchronized unicast transmission	1090
Synchronized unicast transmission	120
Unicast TX to awake receiver	96

Figure 2: Power consumption of atom operations on the Tmote Sky with ContikiMAC

3. MAC and Radio Duty Cycling

In low-power wireless networks, communication and power consumption are intertwined. The communication device is typically the most power-consuming component, but merely refraining from transmissions is not enough to attain a low power consumption: the radio consumes as much power in listen mode as when actively transmitting, as show in Figure 2 . To reduce power consumption, the radio must be switched completely off -- duty-cycled -- as much as possible. ContikiMAC is a very typical Radio Duty Cycling protocol [ContikiMAC].

From the perspective of MAC&RDC, all upper layer protocols, such as routing, RESTful communication, adaption, and management flows, are all applications. Since the duty cycling algorithm is the key to energy-efficiency of the wireless medium, it synchronizes the TX/RX request from the higher layer.

The MAC&RDC are not in the scope of the IETF, yet lower layer designers and chipset manufactures take great care of the problem. For the IETF protocol designers, however, it is good to know the behaviors of lower layers so that the designed protocols can work perfectly with them.

Once again, the IETF protocols we are going to talk about in the following sections are the customers of the lower layer. If they want to get better service in a cooperative way, they should be considerative and understand each other.

3.1. Power Save Services Provided by IEEE 802.11v

IEEE 802.11v defines mechanisms and services for power save of stations/nodes that include flexible multicast service (FMS), proxy ARP advertisement, extended sleep modes, traffic filtering. It would be useful if upper layer protocols knows such capabilities provided by the lower layer, so that they can coordinate with each other.

These services include:

Proxy ARP: The Proxy ARP capability enables an AP to indicate that the non-AP STA will not receive ARP frames. The Proxy ARP capability enables the non-AP STA to remain in power-save for longer periods of time.

BSS Max Idle Period management enables an AP to indicate a time period during which the AP does not disassociate a STA due to non-receipt of frames from the STA. This supports improved STA power saving and AP resource management.

FMS: A service in which a non-access point (non-AP) station (STA) can request a multicast delivery interval longer than the delivery

traffic indication message (DTIM) interval for the purposes of lengthening the period of time a STA may be in a power save state.

Traffic Filtering Service (TFS): A service provided by an access point (AP) to a non-AP station (STA) that can reduce the number of frames sent to the non-AP STA by not forwarding individually addressed frames addressed to the non-AP STA that do not match traffic filters specified by the non-AP STA.

Using the above services provided by the lower layer, the constrained nodes can achieve either client initiated power save (via TFS) or network assisted power save (Proxy-ARP, BSS Max Idle Period and FMS).

Upper layer protocols would better synchronize with the parameters such as FMS interval and BSS MAX Idle Period, so that the wireless transmissions are not triggered periodically.

4. IP Adaption and Transport Layer

6LoWPAN is the adaption layer to run IPv6 over IEEE 802.15.4 MAC&PHY. It was born to fill the gap that the IPv6 layer does not support fragmentation and assembly of <1280-byte packets while IEEE 802.15.4 only supports a MTU of 127 bytes.

IPv6 is the basis for the higher layer protocols, including both TCP/UDP transport and applications. So they are quite ignorant of the transmission and reception behaviors, and are almost neutral to the energy-efficiency problem.

What the network stack can optimize is to save the computing power. For example the Contiki implementation has multiple cross layer optimizations for buffers and energy management, e.g., the computing and validation of UDP/TCP checksums without the need of reading IP headers from a different layer. These optimizations are software implementation techniques, and out of the scope of IETF and the LWIG working group.

The 6LoWPAN contributes to the energy-efficiency problem in two ways. First of all, it swaps computing with communication. 6LoWPAN applies compression of the IPv6 header. This means less amount of data will be handled by the lower layer, but both the sender and receiver should spend more computing power on the compression and decompression of the packets over the air. Secondly, the 6LoWPAN working group developed the energy-efficient Neighbor Discovery called 6LoWPAN-ND, which is an energy efficient replacement of the IPv6 ND in constrained environments. IPv6 Neighbor Discovery was not designed for non-transitive wireless links, as its heavy use of multicast makes it inefficient and sometimes impractical in a low-

power and lossy network. 6LoWPAN-ND describes simple optimizations to IPv6 Neighbor Discovery, its addressing mechanisms, and duplicate address detection for Low-power Wireless Personal Area Networks and similar networks.

5. Routing Protocols

The routing protocol designed by the IETF for constrained environments is called RPL [RFC6550]. As a routing protocol, RPL has to exchange messages periodically and keep routing states for each destination. RPL is optimized for the many-to-one communication pattern, where network nodes primarily send data towards the border router, but has provisions for any-to-any routing as well.

The authors of the Powertrace tool studied the power profile of RPL. It divides the routing protocol into control and data traffic. The control channel uses ICMP messages to establish and maintain the routing states. The data channel is any application that uses RPL for routing packets. The study has shown that the power consumption of the control traffic goes down over time and data traffic stays relatively constant. The study also reflects that the routing protocol should keep the control traffic as low as possible to make it energy-friendly.

6. Application Layer

CoAP [I-D.ietf-core-coap] was designed as a RESTful application protocol, connecting the services of smart devices to the World Wide Web. CoAP is not a chatty protocol, it provides basic communication services such as service discovery and GET/POST/PUT/DELETE methods with a binary header.

The energy-efficient design is implicitly included in the CoAP protocol design. To reduce regular and frequent queries of the resources, CoAP provides an observe mode, in which the requester registers its interest of a certain resource and the responder will report the value whenever it was updated. This reduces the request response roundtrip while keeping information exchange a ubiquitous service.

7. Cross Layer Optimization

The cross layer optimization is a technique used in many scenarios. There are some technologies for power efficient optimization via PHY to Routing cross layer design [Cross-layer-Optimization]. In this research, cross-layer optimization frameworks have been developed to minimize the total power consumption or to maximize the utility-power tradeoff using cooperative diversity.

Also a cross-layer design in multihop wireless networks is proposed for congestion control, routing and scheduling in transport, network and link layers into a coherent framework [Cross-layer-design]. This method and thinking could be applied to the implementation of energy effective cross layer design.

8. Summary

We find a summary section necessary although most IETF documents do not contain it. The points we would like to summarize are as follows.

- a. All Internet protocols, which are in the scope of the IETF, are customers of the lower layers (PHY, MAC, and Duty-cycling). In order to get a better service, the designers of higher layers should know them better.
- b. The IETF has developed multiple protocols for constrained networked devices. A lot of implicitly included design principles have been used in these protocols.
- c. The power trace analysis of different protocol operations showed that for radio-duty-cycled networks broadcasts should be avoided. Saving unnecessary states maintenance is also an effective method to be energy-friendly.

9. IANA Considerations

This document has no IANA requests.

10. Security Considerations

This document discusses the energy efficient protocol design, and does not incur any changes or challenges on security issues besides what the protocol specifications have analyzed.

11. References

11.1. Normative References

[Announcementlayer]

Dunkels, A., "The Announcement Layer: Beacon Coordination for the Sensornet Stack. In Proceedings of EWSN 2011", .

[ContikiMAC]

Dunkels, A., "The ContikiMAC Radio Duty Cycling Protocol, SICS Technical Report T2011:13", December 2011.

[Cross-layer-Optimization]

Le, . and . Hossain, "Cross-Layer Optimization Frameworks for Multihop Wireless Networks Using Cooperative Diversity", July 2008.

[Cross-layer-design]

Chen, ., Low, ., and . Doyle, "Cross-layer design in multihop wireless networks", 2011.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

[I-D.kovatsch-lwig-class1-coap]

Kovatsch, M., "Implementing CoAP for Class 1 Devices", draft-kovatsch-lwig-class1-coap-00 (work in progress), October 2012.

[Powertrace]

Dunkels, ., Eriksson, ., Finne, ., and . Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks", March 2011.

11.2. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.

[RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R.

Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.

[RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

Authors' Addresses

Zhen Cao (Ed.)
China Mobile
Xuanwumenxi Ave. No.32
Beijing 100871
P.R.China

Email: zehn.cao@gmail.com, caozhen@chinamobile.com

Xuan He
Hitachi (China) Research and Development Corporation
301, Tower C North, Raycom, 2 Kexuyuan Nanlu, Haidian District
Beijing 100190
P.R.China

Email: xhe@hitachi.cn

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich, CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

Hui Tian
China Academy of Telecommunication Research
Huayuanbeilu No.52
Beijing, Haidian District 100191
China

Email: tianhui@mail.ritt.com.cn

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

Y. Hong
ETRI
J. Youn
DONG-EUI Univ.
July 15, 2013

Problem statement and Use cases of Sleepy node in Constrained node
networks
draft-hong-lwig-sleepynode-problem-statement-00

Abstract

This document describes the use cases of communication considering sleepy nodes and the problems of connecting to sleepy nodes in constrained node networks. The use cases of communications between sleepy nodes and non-sleepy nodes are classified by the end-to-end communication and the network topology. The adopt of power saving in constrained nodes raises compelling problems in network layer/transport/application layer. In this document, problems of each layer in a sleepy node are described.

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 16, 2014.

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	2
2. Conventions and Terminology	3
3. Related Work and Background	3
4. Use Cases of communication of a sleepy node	4
4.1. Communication between a sleepy node and a non-sleepy node	4
4.2. Communication between sleepy nodes	5
4.3. Communication across routers	5
4.4. Communication within a router	6
4.5. Communication in ad-hoc network	6
5. Problem statement of communication of a sleepy node	6
5.1. Problems of a sleepy node in Network layer	6
5.2. Problems of a sleepy node in Transport layer	7
5.3. Problems of a sleepy node in Application layer	8
6. Security Considerations	8
7. IANA Considerations	8
8. References	8
8.1. Normative References	8
8.2. Informative References	8
Authors' Addresses	9

1. Introduction

Until now, it seems that the Internet protocol (e.g., TCP/IP protocol suite) is not directly related to power management, because it is assumed that network nodes are always connected to main-power. Even though, the network nodes are moving and the network nodes are not connected to main power (e.g., the network nodes may use battery or energy harvesting), the power management has been focused on PHY/MAC layer. Recently, as constrained nodes in constrained node networks become connected to the Internet, it is required to consider power management in Internet protocol in the IETF scape.

The goals for power management may be different by the conditions of device and environment. The general strategies for power management of various conditions are depicted as always-on, always-off, and low-power [I-D.arkko-lwig-cellular]. A constrained node, creating constrained node networks, may occasionally go into sleep mode according to strategies of using power for communication [I-D.ietf-lwig-terminology]. In [I-D.ietf-lwig-terminology], a device is divided into four classes according to energy limitation of

a device. Here, the constrained nodes classed such as class E1 and E2 in classes of energy limitation may occasionally go into sleepy mode. Thus, in constrained node networks, there can exist the end-to-end communications between a sleep mode node and a non-sleep mode node.

Some Internet protocols in the IETF scope assume that the state of a node is always-on mode, such as a non-sleep mode, of a node. While in constrained node networks the state of a node can be divided into a non-sleep mode and sleep mode. Thus, at end-to-end communication perspective, a sleepy node make various problems when Neighbor Discovery is operated and message or data is transmitted between constrained nodes through Internet protocols in the IETF. In particular, because the operation of Neighbor Discovery is done with the assumption that the network node is always-on connected, the operation of Neighbor Discovery of sleepy node may make operational problem. And, sleepy node may affect the performance negatively at application layer and transport layer. First at all, in end-to-end communication perspective, sleepy node can generate unnecessary message/data transmission at application layer and transport layer. In other word, without the state information of a destination node, a source node transmits the message or data to a destination node that goes into sleep mode. This point will affect a constrained node with the limit power negatively in terms of energy efficient of a constrained node.

2. Conventions and Terminology

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

3. Related Work and Background

Power saving in wireless networks is mainly accomplished in PHY/MAC layer. The basic idea of power saving in PHY/MAC layer is to minimize the time of transmission/receipt. In IEEE 802.11 WLAN, the feature of power saving is Power Save Mode (PSM) that is available for nodes existing in an infrastructure based IEEE 802.11 WLAN. PSM is based on a synchronous sleep scheduling policy, in which wireless nodes are able to alternate between an active mode and a sleep mode [PowerMgmt].

Recently, the consideration of power saving is moved to network layer, too. In 6lowpan, the Neighbor Discovery operations must consider the low-power wireless personal area networks such as IEEE 802.15.4. Because the usage of multicast signaling raises severe energy consumption, the Neighbor Discovery optimization for 6lowpan has limited the usage of multicast signaling [I-D.ietf-6lowpan-nd].

In application layer, the CoAP has two functions such as proxy and cache. The proxy function in the CoAP can cache and service requests for sleepy servers. Thus, a client sends a CoAP request to a proxy on behalf of an origin server of sleep mode and then it respond directly to the client through the proxy. Otherwise if the proxy has an invalid representation of the resource in its cache, the proxy has to attempt to get the valid resource from an origin server of sleep mode. The attempt may or may not be successful according to the state of the origin server [I-D.ietf-core-coap].

4. Use Cases of communication of a sleepy node

To describe the problem of sleepy node in constrained node networks, this clause describes the use cases of communication of sleepy nodes. The use case of communication of sleepy nodes looks like that of normal Internet nodes. The difference from the communication between normal Internet nodes is that there are definitely different two types of network nodes : sleepy node and non-sleepy nodes [I-D.ietf-lwig-terminology]. So, the communication of sleepy nodes can be classified into communication between sleepy node and non-sleepy node and communication between sleepy nodes. And by the topology of networks, the communication can be classified into communication across router, communication within a router, and communication in ad-hoc network.

4.1. Communication between a sleepy node and a non-sleepy node

This use case shows the communication between one network node with always-on network connectivity (non-sleepy node) and one network node with sleepy node network capabilities. In this use case, to provide the communication between a non-sleepy node and a sleepy node, it may require new purpose server such as a Proxy server to handle the request of different node with different capabilities. In this case, the new purpose server will be located at the interface of network [I-D.jeong-eman-network-proxy-protocol].

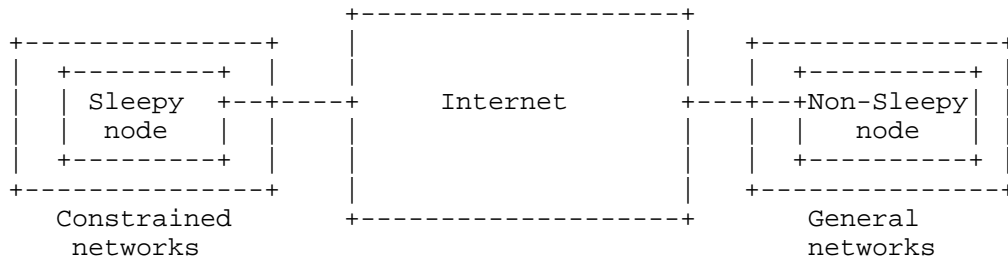


Figure 1: Communication between sleepy node and non-sleepy node

4.2. Communication between sleepy nodes

This use case shows the communication between sleepy nodes within a router. In this use case, the sleepy nodes may use same power saving mechanism and energy efficient technique. And, in this use case, the layer 3 entities such as routers and gateways may benefit from the layer 2 entities such as Access Point and Base Station to keep synchronous sleep scheduling.

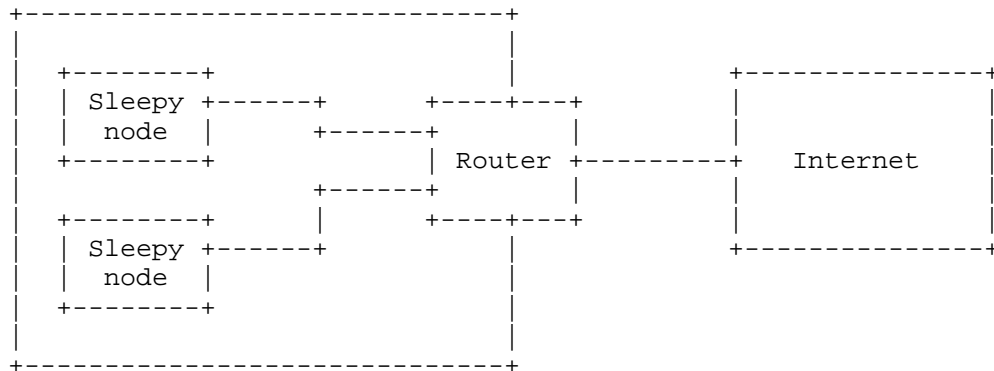


Figure 2: Communication between sleepy nodes

4.3. Communication across routers

This use case is similar to clause 3.1 and multiple hop IP communication is operated. Because each network may have its own power saving and energy efficient mechanism, it is difficult to provide end-to-end level power saving and efficient mechanism.

4.4. Communication within a router

This use is similar to clause 3.2 and 1 hop IP communication is mainly operated. Because it is operated within a same router, it will be possible to provide efficient power saving mechanisms.

4.5. Communication in ad-hoc network

This use case can be happen when it is impossible to implement an infrastructure based wireless network and an infra-less wireless network is constructed. Although there is no explicit a router, it may be possible to select a master and slaves and make the selected master do as a router to provide power saving mechanisms between sleepy nodes. The efficiency of infra-less power saving may be worse than infrastructure-based power saving mechanism. So, it may require to hybrid the infra-less power saving and infrastructure-based power saving.

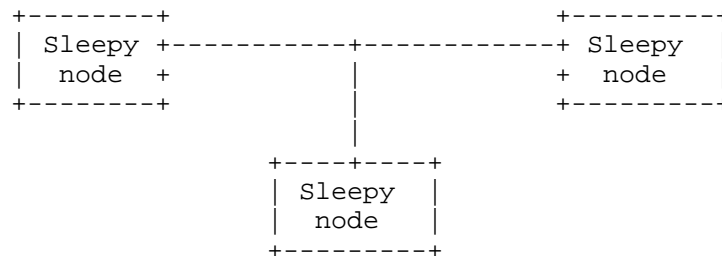


Figure 3: Communication in ad-hoc network

5. Problem statement of communication of a sleepy node

5.1. Problems of a sleepy node in Network layer

The main problems of a sleepy node are related to neighbor discovery [RFC4861]. Among neighbor discovery operations, the operations related to multicast signaling affects severely energy efficiency. So, in 6lowpan-nd, the usage of multicast Neighbor Discovery messages has the limitation with some exception such as initial set of default routers. Because of the limitation of multicast Neighbor Discovery messages, new address assignment with Address Registration Option (ARO) and different Duplicate Address Detection (DAD) mechanisms are used in 6lowpan-nd [I-D.ietf-6lowpan-nd].

Another problem of a sleepy node in network layer is the choosing the proper a sleep time appropriate for its energy characteristics. Even

though the corresponding node is not down or the route path to the corresponding node is not broken, an inappropriate sleep time leads wrong operations. If the modification of Neighbor Discovery such as 6lowpan-nd is applied to network layer in all network nodes, the problems of a sleepy node in network layer may be decreased. In realistic, this modification of Neighbor Discovery such as 6lowpan-nd seems that impossible because there are various wireless networks.

5.2. Problems of a sleepy node in Transport layer

The most constrained environments consist of interoperable IP-capable devices. Thus, a constrained node needs UDP/TCP of transport layer for an end-to-end data transmission service. However a constrained node, creating constrained node networks, is a small device such as sensor device with limited CPU, memory, and power resources. In addition, implementing the whole functionality of the UDP/TCP in a small device becomes a burden. Thus, constrained nodes must implement a minimal functionality for transport service demanded by application in constrained node.

In [I-D.ietf-lwig-guidance], light-weight implementation of transport layer must be considered in a constrained node. Also, the analysis of functionality of the transport protocol is needed to support an end-to-end communication between a non-sleepy node and a sleepy node.

As transport protocol in constrained node with the limit memory, UDP has many advantages. In particular, UDP has a very low overhead for both header size and protocol procedure. This means that UDP will be used as the transport protocol of constrained node in constrained node networks because the packet transmissions and receptions consume less energy and the small space of memory for UDP is needed. This is, the reason is that the CoAP uses UDP as transport protocol to transmit the message of the CoAP. Nevertheless, UDP has drawback that is UDP does not provide any recovery mechanism for lost data. Also, UDP does not have any functionality to support a sleepy node. Thus, source node does not know that data may or may not be successful to the destination node.

TCP has more overhead for both header size and protocol procedure than UDP to be implemented as transport protocol in constrained node. Thus, TCP implementation with the whole functionality in a small device occurs several compelling problems. And it also is not aware to the sleep mode of destination node. TCP protocol is a complex protocol that has a reliable mechanism and the mechanisms such as the sliding window algorithm and congestion control for high throughput. However, the core of TCP is quite simple because many of the complex mechanisms are to improve high-throughput performance. Thus, because high throughput is not a requirement of any end-to-end communication

in most constrained environments, several mechanisms in TCP are not needed TCP mechanism, such as the sliding window algorithm and congestion control, for high throughput. As UDP, TCP also does not have any functionality to support a sleepy node. This is, TCP mistakes data loss generated by sleepy node as data loss over end-to-end transmission. Thus. TCP can perform unnecessary retransmission. This situation can occur in most constrained environments.

5.3. Problems of a sleepy node in Application layer

As the CoAP, it is up to the application to support sleepy node to end-to-end transport service, which also increases the complexity in constrained node. Also there is still no standard transport protocol that can support sleepy node. Thus, to support an end-to-end transport service between a sleep mode node and a non-sleep mode node, the analysis of transport protocols is needed.

6. Security Considerations

TBD.

7. IANA Considerations

TBD

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

8.2. Informative References

- [I-D.arkko-lwig-cellular]
Arkko, J., Eriksson, A., and A. Keranen, "Building Power-Efficient CoAP Devices for Cellular Networks", ID draft-arkko-lwig-cellular-00, February 2013.
- [I-D.ietf-6lowpan-nd]
Shelby, Z., Chakrabartiy, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", ID draft-ietf-6lowpan-nd-21, August 2012.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", ID draft-ietf-core-coap-18, June 2013.

[I-D.ietf-lwig-guidance]

Bormann, C., "Guidance for Light-Weight Implementations of the Internet Protocol Suite ", ID draft-ietf-lwig-guidance-03, February 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", ID draft-ietf-lwig-terminology-05, July 2013.

[I-D.jeong-eman-network-proxy-protocol]

Jeong, S., "Network Proxy Protocol", ID draft-jeong-eman-network-proxy-protocol-01, February 2013.

[PowerMgmt]

Klues, K., "Power Management in Wireless Networks", Report, for Advanced Topics in Networking: Wireless and Mobile Networking by R. Jain, Wash. Univ. in St. Louis, , 2006.

Authors' Addresses

Yong-Geun Hong
ETRI
218 Gajeong-ro Yuseung-Gu
Daejeon 305-700
Korea

Phone: +82 42 860 6557
Email: yghong@etri.re.kr

JooSang Youn
DONG-EUI Univ.
Busan
Korea

Phone: +82 51 890 1993
Email: joosang.youn@gmail.com

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

M. Kovatsch
ETH Zurich
O. Bergmann
Universitaet Bremen TZI
E. Dijk
Philips Research
X. He
Hitachi (China) R&D Corp.
C. Bormann, Ed.
Universitaet Bremen TZI
July 15, 2013

CoAP Implementation Guidance
draft-kovatsch-lwig-coap-01

Abstract

The Constrained Application Protocol (CoAP) is designed for resource-constrained nodes and networks, e.g., sensor nodes in a low-power lossy network (LLN). Yet to implement this Internet protocol on Class 1 devices (i.e., ~ 10 KiB of RAM and ~ 100 KiB of ROM) also lightweight implementation techniques are necessary. This document provides lessons learned from implementing CoAP for tiny, battery-operated networked embedded systems. The guidelines for transmission state management and developer APIs can also help with the implementation of CoAP for less constrained nodes.

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 16, 2014.

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	2
2. Message Processing	3
2.1. Message Buffer Usage	4
2.2. Header Option Parsing and Management	5
2.2.1. On-the-fly Processing	5
2.2.2. Internal Data Structure	5
2.3. Retransmissions	6
2.4. Deduplication	7
2.4.1. Managing Peer MIDs	7
2.4.2. Resource-specific Deduplication	10
3. Transmission State Management	10
3.1. Request/Response Layer	10
3.2. Message Layer	11
4. Observing	12
5. Block-wise Transfers	13
6. Application Developer API	13
7. References	14
7.1. Normative References	14
7.2. Informative References	14
Authors' Addresses	15

1. Introduction

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 against high latency induced by small bandwidth shares or slow request processing in end nodes. To leverage integration of constrained nodes with the world-wide Internet, the protocol design was led by the REST architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [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 unavoidable for the core protocol and thus must be implemented on every node has been kept very small, minimizing the unnecessary accumulation of "optional" features. Options that - when present - are critical for message processing are explicitly marked as such to force immediate rejection of messages with unknown critical options. Third, the syntax of protocol data units is easy to parse and is carefully defined to avoid creation of state in servers where possible.

Although these features enable lightweight implementations of the Constrained Application Protocol, there is still a tradeoff between robustness and latency of constrained nodes on one hand and resource demands (such as battery consumption, dynamic memory needs, and static code size) on the other. The present document gives some guidance on possible strategies to solve this tradeoff 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].

2. Message Processing

For constrained nodes of Class 1 or even Class 2, the most limiting factors for (wireless) network communication usually are memory size and battery lifetime. Most applications therefore try to minimize internal buffer space for both transmit and receive operations, and to maximize sleeping cycles.

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

2.1. Message Buffer Usage

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

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

Depending on the number of requests that can be handled in parallel, an implementation might create a stub response filled with any option that has to be copied from the original request to the separate response, especially the Token option. The drawback of this technique is that the server must be prepared to receive retransmissions of the previous (Confirmable) request to which a new

acknowledgement must be generated. If memory is an issue, a single buffer can be used for both tasks: Only the message type and code must be updated, changing the message id is optional. Once the resource representation is known, it is added as new payload at the end of the stub response. Acknowledgements still can be sent as described before as long as no additional options are required to describe the payload.

2.2. Header Option Parsing and Management

There are two alternatives to handle the header: Either process the header on the fly when an option is accessed or initially parse all values into an internal data structure.

2.2.1. On-the-fly Processing

The advantage of on-the-fly processing is that the compact encoding saves memory and fully reuses the buffer for incoming messages. The basic message header information should always be copied into an internal data structure, as Message ID and/or Token are required for request/response matching and generating the response. Once the message is accepted for further processing, the set of options contained in the received message must be decoded to check for unknown critical options. To avoid multiple passes through the option list, the option parser might maintain a bit-vector where each bit represents an option number that is present in the received request. With the wide and sparse range of option numbers, the number itself cannot be used to indicate the number of left-shift operations to mask the corresponding bit. Hence, an implementation-specific enum of supported options should be used to mask the present options of a message in the bitmap. In addition, the byte index of every option can be added to a sparse list (e.g., a one-dimensional array) for fast retrieval.

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.)

2.2.2. Internal Data Structure

Using an internal data structure for all parsed options has advantages when processing the values, as they are already in a variable of corresponding type and integers in host byte order. The incoming payload and byte strings of the header can be accessed

directly in its IP buffer using pointers. This approach also benefits from a bitmap. Otherwise special values must be reserved to encode an unset option, which might require a larger type than required for the actual value range (e.g., a 32-bit integer instead of 16-bit).

The byte strings (e.g., the URI) are usually not required when generating the response. Thus, this alternative also facilitates the usage of the IP buffer for message assembly - all important values are copied from the shared incoming/outgoing buffer.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number order required for the delta encoding. The CoAP encoding is then applied in a serialization step before sending. On-the-fly processing might require extensive memmove operations to insert new header options or needs to restrict developers to set options in order.

2.3. Retransmissions

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

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

For platforms that disable clock tick interrupts in sleep states, the application must take into consideration the clock deviation that occurs during sleep (or ensure to remain in idle state until the message has been acknowledged or the maximum number of retransmissions is reached). Since CoAP allows up to four retransmissions with a binary exponential back-off it could take up to 45 seconds until the send operation is complete. Even in idle

state, this means substantial energy consumption for low-power nodes. Implementers therefore might choose a two-step strategy: First, do one or two retransmissions and then, in the later phases of back-off, go to sleep until the next retransmission is due. In the meantime, the node could check for new messages including the acknowledgement for any Confirmable message to send.

2.4. Deduplication

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.

2.4.1. Managing Peer MIDs

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	111101110111	t_1
... etc.		

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

The presence of a table row with base `MID_i` (regardless of the bitfield values) indicates that a value `MID_i` has been received at a time `t_i`. Subsequently, each bitfield bit `k` ($0 \dots K-1$) 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.

2.4.2. Resource-specific Deduplication

Deduplication is heavy for Class 1 devices, as the number of peer addresses can be vast. Servers should be kept stateless, i.e., the REST API should be designed idempotent whenever possible. When this is not the case, the resource handler could perform an optimized deduplication by exploiting knowledge about the application. Another, server-wide strategy is to only keep track of non-idempotent requests.

3. Transmission State Management

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

3.1. Request/Response Layer

Figure 1 depicts the two states at the request/response layer of a CoAP client. When a request is issued, a "reliable_send" or "unreliable_send" is triggered at the message layer. The WAITING state can be left through three transitions: Either the client cancels the request and triggers cancellation of a CON transmission at the message layer, the client receives a failure event from the message layer, or a receive event containing a response.

```

+-----CANCEL-----+
|           / M_CMD(cancel)           |

```

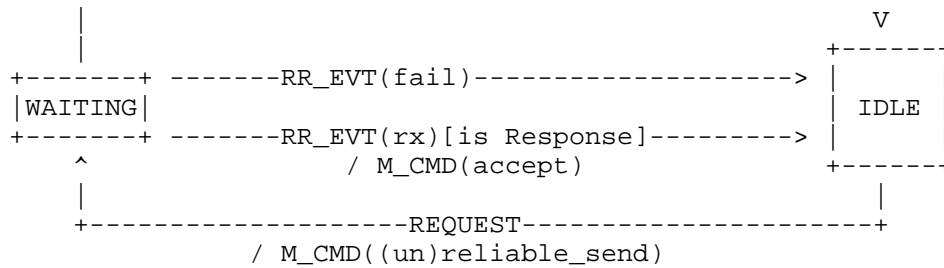


Figure 1: CoAP Client Request/Response Layer FSM

A server resource can decide at the request/response layer whether to respond with a piggy-backed or a separate response. Thus, there are two busy states in Figure 2, SERVING and SEPARATE. An incoming receive event with a NON request directly triggers the transition to the SEPARATE state.

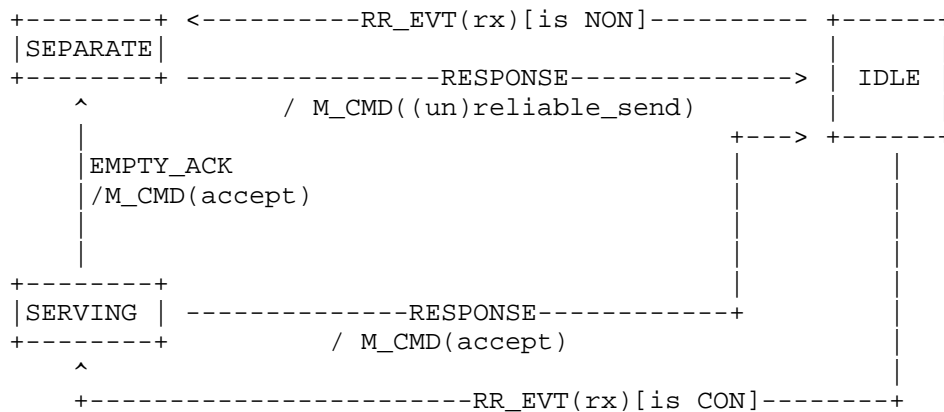


Figure 2: CoAP Server Request/Response Layer FSM

3.2. Message Layer

Figure 3 shows the different states of a CoAP endpoint per message exchange. Besides the linking action `RR_EVT()`, the message layer has a `TX` action to send a message. For sending and receiving NONs, the endpoint remains in its `CLOSED` state. When sending a CON, the endpoint remains in `RELIABLE_TX` and keeps retransmitting until the transmission times out, it receives a matching RST, the request/response layer cancels the transmission, or the endpoint receives an implicit acknowledgement through a matching NON or CON. Whenever the endpoint receives a CON, it transitions into the `ACK_PENDING` state, which can be left by sending the corresponding ACK.

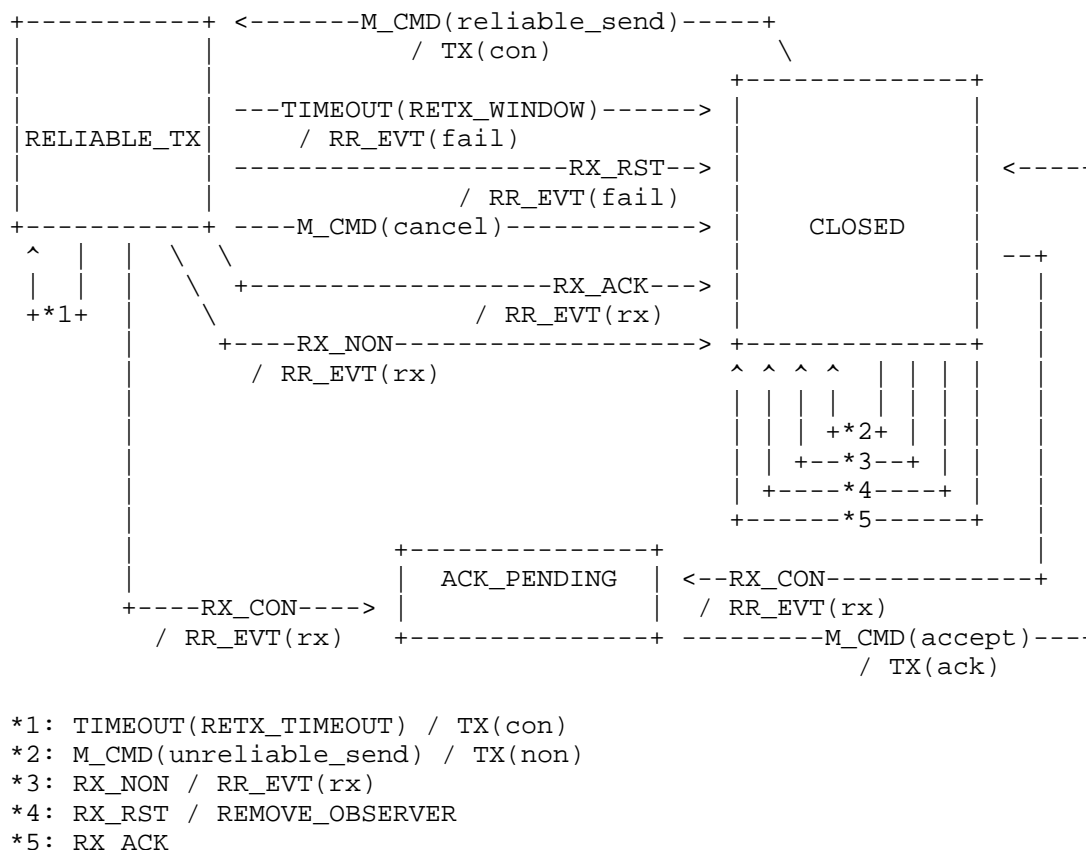


Figure 3: CoAP Message Layer FSM

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

4. Observing

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

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

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

5. Block-wise Transfers

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

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

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

6. Application Developer API

Bringing a Web transfer protocol to constrained environments does not only change the networking of the corresponding systems, but also the way they should be programmed. A CoAP implementation should provide a developer API similar to REST frameworks in traditional computing. A server should not be created around an event loop with several function calls, but rather by implementing handlers following the resource abstraction.

So far, the following types of RESTful resources were identified:

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

PARENT A parent resource manages several sub-resources by programmatically evaluating the Uri-Path, which may be longer than that of the parent resource. Defining a URI templates (see [RFC6570]) would be a convenient way to pre-parse arguments given in the Uri-Path.

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

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

7. References

7.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [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.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

7.2. 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-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

Authors' Addresses

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
CH-8092 Zurich
Switzerland

Email: kovatsch@inf.ethz.ch

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

Email: bergmann@tzi.org

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

Xuan He
Hitachi (China) R&D Corp.
301, Tower C North, Raycom, 2 Kexuyuan Nanlu
Beijing, 100190
P.R.China

Email: xhe@hitachi.cn

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 13, 2014

Y. Ma
Hitachi
Z. Cao
China Mobile
July 12, 2013

Implication of 3GPP Link Characteristics on Lightweight IP Design
draft-ma-lwig-3gpplink-imply-00

Abstract

In 3GPP Release 12, the work item Machine Type Communication (MTC) is specifying low cost terminals for Machine to Machine communications. Since IETF has already developed a suite of protocols for device communication, it is useful to analyze the limitation of 3GPP MTC and the impact on the implementation of IETF protocol suite. This document analyzes the feature of 3GPP MTC and the impact on light weight protocol implementation for the MTC terminals.

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 13, 2014.

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	2
1.1. Conventions used in this document	2
1.2. Terminology	3
2. 3GPP MTC features	3
3. Impact on light weight implementation	4
3.1. Network layer	4
3.2. Transport Layer	4
3.3. Application Layer	5
4. IANA Considerations	5
5. Security Considerations	5
6. References	5
6.1. Normative References	5
6.2. Informative References	6
Authors' Addresses	6

1. Introduction

As the Internet of Things are booming, it is important for cellular networks to support the Machine to Machine communication. In 3GPP one work item is set up since Release 10 to deal with the so called Machine Type Communication (MTC) in cellular network.

At the same time, IETF has developed a suite of Internet protocols suitable for small devices, including 6LowPAN [RFC6282], 6LowPAN-ND [RFC6775], RPL[RFC6550], COAP[I-D.ietf-core-coap].

This document tries to summarize the feature of 3GPP MTC and the impact on implementation of the IETF light-weight protocols suite in each layer. Link characteristic implications on upper layer protocols are also analyzed in [I-D.hex-lwig-energy-efficient].

1.1. Conventions used in this document

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

1.2. Terminology

The terminologies used in this document can be referred to [I-D.ietf-lwig-terminology].

2. 3GPP MTC features

In 3GPP the scenarios for MTC and the features of the MTC is analyzed. Following are the features which is considered specific for MTC.[TS22.368]

1. Low Mobility: It is intended for use with MTC Devices that do not move, move infrequently, or move only within a certain region.
2. Time Controlled: It is intended for use with MTC Applications that can tolerate to send or receive data only during defined time intervals and avoid unnecessary signalling outside these defined time intervals.
3. Small Data Transmissions: It is intended for use with MTC Devices that send or receive small amounts of data. The observed size of many of the instances of data exchanges is on the order of 1K (1024) octets.
4. Infrequent Mobile Terminated: It is intended for use with MTC Devices that mainly utilize mobile originated communications.
5. MTC Monitoring: It is intended for monitoring MTC Device related events.
6. Secure Connection: It is intended for use with MTC Devices that require a secure connection between the MTC Device and MTC Server /MTC Application Server.
7. Group Based MTC Features: It is a MTC Feature that applies to a MTC Group. Generally the system shall be optimized to handle MTC Groups. The system shall provide a mechanism to associate an MTC Device to a single MTC Group. There are two sub features for Group based MTC features.
 - a. Group Based Policing is intended for use with a MTC Group for which the network operator wants to enforce a combined QoS policy.
 - b. Group Based Addressing is intended for use with a MTC Group for which the network operator wants to optimize the message volume when many MTC Devices need to receive the same message.

On the other hand, IETF has developed multiple protocols to enable end-to-end IP communication between constrained nodes and fully capable nodes. These works have witnessed the evolution of the traditional Internet protocol stack to the light-weight Internet protocol stack. As show in the below , IETF has developed CoAP as the application layer, and 6LowPAN as the adaption layer to run IPv6 on IEEE 802.15.4 and Bluetooth Low-Energy, with the support of routing from RPL and efficient neighbor discovery from 6LowPAN-ND.

However according to the features of 3GPP MTC, not all IETF protocol suites are necessary to be implemented for cellular MTC devices. In this document the impact of the MTC features on IETF protocol suites are analysed.

3. Impact on light weight implementation

3.1. Network layer

IPv6 is mandatory for 3GPP terminals. The consideration to implement IPv6 for 3GPP terminal is specified in [RFC3316]. As specified in RFC3316, standard IPv6 protocol stack is used. The MTC device does not need to support 6lowpan, 6lowpan-ND, or RPL for the communication between MTC device and MTC server or application in the cellular network.

However in cellular network deployment scenario, the MTC device is sometimes used as gateway device to bridge other resource constrained nodes to the cellular network. So the MTC device needs to implement 6lowpan-ND or RPL if other resource constrained nodes is connected through 802.15.4 or other wireless technologies.

3.2. Transport Layer

For MTC device, the assumed use case and scenario is about small data transmission. And the reliability is not required. Therefore the UDP based transport is suitable for MTC and should be mandatory for all cellular terminals.

3GPP MTC needs to support secure connection between MTC device and MTC server/ MTC Application Server. In this case the light-weight secure transport protocol such as DTLS should be supported by 3GPP MTC device.

3.3. Application Layer

CoAP [I-D.ietf-core-coap] was designed as a restful application protocol, connecting the smart devices application and service to the world-wide-web. It provides basic communication services such as service discovery and GET/POST/PUT/DELETE methods with a binary header. It is assumed to work over IPv6 in the network layer.

Although IPv6 is made mandatory for 3GPP terminals, IPv4 is still considered a default protocol for 3GPP MTC [TS23.888]. IPv4 based communication is used for communication between MTC device and MTC server. Many CoAP features are based on IPv6 assumption. For example, auto-configuration, resource discovery, etc. Therefore if CoAP is used for cellular network, it is necessary to consider how to support IPv4 for the mentioned CoAP features.

4. IANA Considerations

This document has no IANA requests.

5. Security Considerations

The security implementation should follow both 3GPP and IETF specifications.

6. References

6.1. Normative References

[I-D.hex-lwig-energy-efficient]

Cao, Z., He, X., and M. Kovatsch, "Energy Efficient Implementation of IETF Protocols on Constrained Devices", draft-hex-lwig-energy-efficient-00 (work in progress), February 2013.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

[I-D.kovatsch-lwig-class1-coap]

Kovatsch, M., "Implementing CoAP for Class 1 Devices",
draft-kovatsch-lwig-class1-coap-00 (work in progress),
October 2012.

[TS22.368]

3GPP, "TS 22.368: Service requirements for Machine-Type
Communications", March 2013,
<<http://www.3gpp.org/ftp/Specs/html-info/22368.htm>>.

[TS23.888]

3GPP, "TS 23.888: System improvements for Machine-Type
Communications", September 2012,
<<http://www.3gpp.org/ftp/Specs/html-info/23888.htm>>.

6.2. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3316] Arkko, J., Kuijpers, G., Soliman, H., Loughney, J., and J.
Wiljakka, "Internet Protocol Version 6 (IPv6) for Some
Second and Third Generation Cellular Hosts", RFC 3316,
April 2003.
- [RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6
Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
September 2011.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R.,
Levis, P., Pister, K., Struik, R., Vasseur, JP., and R.
Alexander, "RPL: IPv6 Routing Protocol for Low-Power and
Lossy Networks", RFC 6550, March 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann,
"Neighbor Discovery Optimization for IPv6 over Low-Power
Wireless Personal Area Networks (6LoWPANs)", RFC 6775,
November 2012.

Authors' Addresses

Yuanchen Ma
Hitachi
301, Tower C North, Raycom, 2 Kexuyuan Nanlu, Haidian District
Beijing 100190
P.R.China

Email: ycma@hitachi.cn

Zhen Cao
China Mobile
Xuanwumenxi Ave. No.32
Beijing 100871
P.R.China

Email: zehn.cao@gmail.com, caozhen@chinamobile.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2014

H. Tschofenig
Nokia Siemens Networks
S.S. Kumar
S. Keoh
Philips Research
July 5, 2013

A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol
for Smart Objects and Constrained Node Networks
draft-tschofenig-lwig-tls-minimal-03

Abstract

Transport Layer Security (TLS) is a widely used security protocol that offers communication security services at the transport layer. The initial design of TLS was focused on the protection of applications running on top of the Transmission Control Protocol (TCP), and was a good match for securing the Hypertext Transfer Protocol (HTTP). Subsequent standardization efforts lead to the publication of the Datagram Transport Layer Security (DTLS) protocol, which allows the re-use of the TLS security functionality and the payloads to be exchanged on top of the User Datagram Protocol (UDP).

With the work on the Constrained Application Protocol (CoAP), as a specialized web transfer protocol for use with constrained nodes and constrained networks, DTLS is a preferred communication security protocol.

Smart objects are constrained in various ways (e.g., CPU, memory, power consumption) and these limitations may impose restrictions on the protocol stack such a device runs. This document only looks at the security part of that protocol stacks and the ability to customize TLS/DTLS. To offer input for implementers and system architects this document illustrates the costs and benefits of various TLS/DTLS features for use with smart objects and constraint node networks.

Requirements Language

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

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
2. Overview	4
3. Design Decisions	6
4. Performance Numbers	7
4.1. Pre-Shared Key (PSK) based DTLS implementation	7
4.1.1. Prototype Environment	7
4.1.2. Code size and Memory Consumption	8
4.1.3. Communication Overhead	8
4.1.4. Message Delay, Success Rate and Bandwidth	9
4.2. Certificate based and Raw-public key based TLS implementation	10
4.3.1. Prototype Environment	10
4.3.2. Code size	10
4.3.2. Raw Public Key Implementation	11
5. Summary and Conclusions	12
6. Security Considerations	12
7. IANA Considerations	13

8. Acknowledgements	13
9. References	13
9.1. Normative References	13
9.2. Informative References	13
Authors' Addresses	15

1. Introduction

The IETF published three versions of Transport Layer Security: TLS Version 1.0 [RFC2246], TLS Version 1.1 [RFC4346], and TLS Version 1.2 [RFC5246]. Section 1.1 of [RFC4346] explains the differences between Version 1.0 and Version 1.1; those are small security improvements, including the usage of an explicit initialization vector to protect against cipher-block-chaining attacks, which all have little to no impact on smart object implementations. Section 1.2 of [RFC5246] describes the differences between Version 1.1 and Version 1.2. TLS 1.2 introduces a couple of major changes with impact to size of an implementation. In particular, prior TLS versions hard-coded the MD5/SHA-1 combination in the pseudo-random function (PRF). As a consequence, any TLS Version 1.0 and Version 1.1 implementation had to have MD5 and SHA-1 code even if the remaining cryptographic primitives used other algorithms. With TLS Version 1.2 the two had been replaced with cipher-suite-specified PRFs. In addition, the TLS extensions definition [RFC6066] and various AES ciphersuites [RFC3268] got merged into the TLS Version 1.2 specification.

All three TLS specifications list a mandatory-to-implement ciphersuite: for TLS Version 1.0 this was `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA`, for TLS Version 1.1 it was `TLS_RSA_WITH_3DES_EDE_CBC_SHA`, and for TLS Version 1.2 it is `TLS_RSA_WITH_AES_128_CBC_SHA`. There is, however, an important qualification to these compliance statements, namely that they are only valid in the absence of an application profile standard specifying otherwise. The smart object environment may, for example, represent a situation for such an application profile which defines a cryptosuite that reduces memory and computation requirements without sacrificing security.

All TLS versions offer a separation between authentication and key exchange, and bulk data protection. The former is more costly performance- and message-wise. The details of the authentication and key exchange, using the TLS Handshake, vary with the chosen ciphersuite. With new ciphersuites the TLS feature-set can easily be enhanced, in case the already large collection of ciphersuites, see [TLS-IANA], does not match the requirements.

Once the TLS Handshake has been successfully completed the necessary keying material and parameters are setup for usage with the TLS Record Layer, which is responsible for bulk data protection. The provided security of the TLS Record Layer depends also, but not only, on the chosen ciphersuite algorithms; NULL encryption ciphersuites, like those specified in RFC 4785 [RFC4785], offer only integrity-without confidentiality-protection. Example ciphersuites for the TLS Record Layer are RC4 with SHA-1, AES-128 with SHA-1, AES-256 with SHA-1, RC4 with SHA-1, RC4 with MD5. It is worth mentioning that TLS may also be used without the TLS Record Layer. This has, for example, been exercised with the work on the framework for establishing a Secure Real-time Transport Protocol (SRTP) security context using the Datagram Transport Layer Security (DTLS [RFC4347]) protocol (DTLS-SRTP [RFC5763]).

It is fair to say that TLS and consequently DTLS offers a fair degree of flexibility. What specific security features of TLS are required for a specific smart object application scenario depends on various factors, including the communication architecture and the threats that shall be mitigated.

The goal of this document is to provide guidance on how to use existing DTLS/TLS extensions for smart objects and to explain their costs in terms of code size, computational effort, communication overhead, and (maybe) energy consumption. The document does not try to be exhaustive, as the list of TLS/DTLS extensions is enhanced on a frequent basis. Instead we focus on extensions that those working in the smart object community often found valuable in their practical experience. A non-goal is to propose new extensions to DTLS/TLS to provide even better performance characteristics in specific environments.

2. Overview

A security solution to be deployed is strongly influenced by the communication relationships [RFC4101] between the entities. Having a good understanding of these relationships will be essential to define the threats and decide on how to customize the security solution. Some of these considerations are described in [I-D.gilger-smart-object-security-workshop].

Consider the following scenario where a smart-meter transmits its energy readings to other parties. The public utility has to ensure that the meter readings it obtained can be attributed to a specific meter in a household. It is simply not acceptable for public utility to have any meter readings tampered in transit or by a rogue endpoint (particularly if the attack leads to a disadvantage, for example financial loss, for the utility). Users in a household may want to

ensure that only certain authorized parties are able to read their meter; privacy concerns come to mind.

In this example, a smart-meter may only ever need to talk to servers of a specific utility or even only to a single pre-configured server. Clearly, some information has to be pre-provisioned into the device to ensure the desired behavior to talk only to selected servers. The meter may come pre-configured with the domain name and certificate belonging to the utility. The device may, however, also be configured to accept one or multiple server certificates. It may even be pre-provisioned with the server's raw public key, or a shared secret instead of relying on certificates.

Lowering the flexibility decreases the implementation overhead. If shared secrets are used with TLS-PSK [RFC4279] or raw public keys are used with TLS [I-D.ietf-tls-oob-pubkey], fewer lines of code are needed than employing X.509 certificate, as will be explained later in this document. A decision for constraining the client-side TLS implementation, for example by offering only a single ciphersuite, has to be made in awareness of what functionality will be available on the TLS server-side. In certain communication environments it may be easy to influence both communication partners while in other cases the existing deployment needs to be taken into consideration.

To illustrate another example, consider an Internet radio, which allows a user to connect to available radio stations. A device like this will be more demanding than an IP-enabled weighing scale that only connects to the single web server offered by the device manufacturer. A threat assessment may even lead to the conclusion that TLS support is not necessary at all in this particular case.

Consider the following extension to our earlier scenario where the smart-meter is attached to a home WLAN network and the inter-working with WLAN security mechanisms need to be taken care of. On top of the link layer authentication, a transport layer or application layer security mechanism needs to be implemented. Quite likely the security mechanisms will be different due to the different credential requirements. While there is a possibility for re-use of cryptographic libraries (such as the SHA-1, MD5, or HMAC) the overall code footprint will very likely be larger.

Furthermore, security technology that will be deployed by end-user consumer market products and large enterprise customer products will need to be customized completely different. While the security building blocks may be reused, there is certainly a big difference between in terms of the architecture, the threats and effort that will be spent securing the system.

3. Design Decisions

To evaluate the required TLS functionality a couple of high level design decisions have to be made:

- o What type of protection for the data traffic is required? Is confidentiality protection in addition to integrity protection required? Many TLS ciphersuites also provide a variant for NULL encryption [RFC4279]. If confidentiality protection is required, a carefully chosen set of algorithms may have a positive impact on the code size. Re-use of crypto-libraries (within TLS but also among the entire protocol stack) will also help to reduce the overall code size.
- o What functionality is available in hardware? For example, certain hardware platforms offer support for a random number generator as well as cryptographic algorithms (e.g., AES). These functions can be re-used and allow to reduce the amount of required code. Using hardware support not only reduces the computation time but can also save energy due to the optimized implementation.
- o What credentials for client and server authentication are required: passwords, pre-shared secrets, certificates, raw public keys (or a mixture of them)? Is mutual authentication required? Is X509 certificate handling necessary? If not, then the ASN.1 library as well as the certificate parsing and processing can be omitted. If pre-shared secrets are used then the big integer implementation can be omitted.
- o What TLS version and what TLS features, such as session resumption, can or have to be used? In the case of DTLS, generic fragmentation and reordering requires large buffers to reassemble the messages, which might be too heavy for some devices.
- o Is it possible to design only the client-side TLS stack, or necessary to provide the server-side implementation as well? Handshake messages sent are different sizes for the client and server which creates energy consumption differences (due to the fact that more power is spent during transmission than while receiving data in wireless devices).
- o Which side will be more powerful? Resource-constrained sensor nodes running CoAPS might be server only, while nodes running HTTPS are most like clients only that post their information to a normal Web server. The constrained side will most likely only implement a single ciphersuite. Flexibility is given to a more powerful counterpart that supports many different ciphersuite for various connected devices.

- o Is it possible to hardwire credentials into the code rather than loading them from storage? If so, then no file handling or parsing of the credentials is needed and the credentials are already available in a form that they can be used within the TLS implementation.

4. Performance Numbers

In this section we summarize performance measurements available from certain implementation experiences. This section is not supposed to be exhaustive as we do not have all measurements available. The performances are grouped according to extensions (TLS-PSK, raw-public key and certificate based) and further grouped by performance measures (memory, code size, communication overhead, etc.). Where possible we extract the different building blocks found in TLS and present their performance measures individually.

4.1. Pre-Shared Key (PSK) based DTLS implementation

This section provides performance numbers for a prototype implementation of DTLS-PSK described in [I-D.keoh-lwig-dtls-iot] and evaluates the memory and communication overheads.

4.1.1. Prototype Environment

The prototype is written in C and runs as an application on Contiki OS 2.5 [Dunkels-contiki], an event-driven open source operating system for constrained devices. They were tested in the Cooja simulator and then ported to run on Redbee Econotag hardware, which features a 32-bit CPU, 128 KB of ROM, 128 KB of RAM, and an IEEE 802.15.4 enabled radio with an AES hardware coprocessor. The prototype comprises all necessary functionality to adapt to the roles as a domain manager or a joining device.

The prototype is based on the "TinyDTLS" [Bergmann-Tinydtls] library and includes most of the extensions and the adaptation as follows:

- (1) The cookie mechanism was disabled in order to fit messages to the available packet sizes and hence reducing the total number of messages when performing the DTLS handshake.
- (2) Separate delivery was used instead of flight grouping of messages and redesigned the retransmission mechanism accordingly.
- (3) The "TinyDTLS" AES-CCM module was modified to use the AES hardware coprocessor.

The following subsections further analyze the memory and

communication overhead of the solution.

4.1.2. Code size and Memory Consumption

Table 1 presents the codesize and memory consumption of the prototype differentiating (i) the state machine for the handshake, (ii) the cryptographic primitives, and (iii) the DTLS record layer mechanism.

The use of DTLS appears to incur large memory footprint both in ROM and RAM for two reasons. First, DTLS handshake defines many message types and this adds more complexity to its corresponding state machine. The logic for message re-ordering and retransmission also contributes to the complexity of the DTLS state machine. Second, DTLS uses SHA2-based crypto suites which is not available from the hardware crypto co-processor.

	DTLS	
	ROM	RAM
State Machine	8.15	1.9
Cryptography	3.3	1.5
Key Management	1.0	0.0
DTLS Record Layer	3.7	0.5
TOTAL	16.15	3.9

Table 1: Memory Requirements in KB

4.1.3. Communication Overhead

The communication overhead is evaluated in this section. In particular, the message overhead and the number of exchanged bytes under ideal condition without any packet loss is examined.

Table 2 summarizes the required number of round trips, number of messages and the total exchanged bytes for the DTLS-based handshake carried out in ideal conditions, i.e., in a network without packet losses. DTLS handshake is considered complex as it involves the exchange of 12 messages to complete the handshake. Further, DTLS runs on top the transport layer, i.e., UDP, and hence this directly increases the overhead due to lower layer per-packet protocol headers.

	DTLS
No. of Message	12
No. of round trips	4
802.15.4 headers	168B
6LowPAN headers	480B
UDP headers	96B
Application	487B
TOTAL	1231B

Table 2: Communication overhead for Network Access and Multicast Key Management

4.1.4. Message Delay, Success Rate and Bandwidth

Section 5.3 provided an evaluation of the protocol in an ideal condition, thus establishing the the baseline protocol overhead. The prototype was further examined and simulated the protocol behavior by tuning the packet loss ratio. In particular, the impact of packet loss on message delay, success rate and number of messages exchanged in the handshake were examined.

Figure 4 shows the percentage of successful handshakes as a function of timeouts and packet loss ratios. As expected, a higher packet loss ratio and smaller timeout (15s timeout) result in a failure probability of completing the DTLS handshake. When the packet loss ratio reaches 0.5, practically no DTLS handshake would be successful.

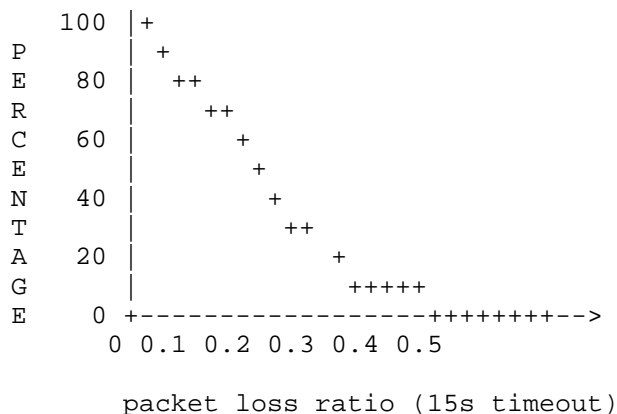


Figure 1: Average % of successful handshakes

Delays in network access and communication are intolerable since they lead to higher resource consumption. As the solution relies on PSK, the handshake protocol only incurs a short delay of a few milliseconds when there is no packet loss. However, as the packet loss ratio increases, the delay in completing the handshake becomes significant because lost packets must be retransmitted. Our implementation shows that with a packet loss ratio of 0.5, the times to perform network access and multicast key management could take up to 24s.

Finally, another important criterion is the number of messages exchanged in the presence of packet loss. A successful handshake could incur up to 35 or more messages to be transmitted when the packet loss ratio reaches 0.5. This is mainly because the DTLS retransmission is complex and often requires re-sending multiple messages even when only a single message has been lost.

4.2. Certificate based and Raw-public key based TLS implementation

4.3.1. Prototype Environment

The following code was compiled under Ubuntu Linux using the -Os compiler flag setting for a 64-bit AMD machine using a modified version of the axTLS embedded SSL implementation.

4.3.2. Code size

For the cryptographic support functions these are the binary sizes:

Cryptographic functions	Code size
MD5	4,856 bytes
SHA1	2,432 bytes
HMAC	2,928 bytes
RSA	3,984 bytes
Big Integer Implementation	8,328 bytes
AES	7,096 bytes
RC4	1,496 bytes
Random Number Generator	4,840 bytes

Table 3: Code-size for cryptographic functions

The TLS library with certificate support consists of the following parts:

x509 related code: 2,776 bytes

The x509 related code provides functions to parse certificates, to

copy them into the program internal data structures and to perform certificate related processing functions, like certificate verification.

ASN1 Parser: 5,512 bytes

The ASN1 library contains the necessary code to parse ASN1 data.

Generic TLS Library: 15,928 bytes

This library is separated from the TLS client specific code to offer those functions that are common with the client and the server-side implementation. This includes code for the master secret generation, certificate validation and identity verification, computing the finished message, ciphersuite related functions, encrypting and decrypting data, sending and receiving TLS messages (e.g., finish message, alert messages, certificate message, session resumption).

TLS Client Library: 4,584 bytes

The TLS client-specific code includes functions that are only executed by the client based on the supported ciphersuites, such as establishing the connection with the TLS server, sending the ClientHello handshake message, parsing the ServerHello handshake message, processing the ServerHelloDone message, sending the ClientKeyExchange message, processing the CertificateRequest message.

OS Wrapper Functions: 2,776 bytes

These functions aim to make development easier (e.g., for failure handling with memory allocation and various header definitions) but are not absolutely necessary.

OpenSSL Wrapper Functions: 931 bytes

The OpenSSL API calls are familiar to many programmers and therefore these wrapper functions are provided to simplify application development. This library is also not absolutely necessary.

Certificate Processing Functions: 4,456 bytes

These functions provide the ability to load certificates from files (or to use a default key as a static data structure embedded during compile time), to parse them, and populate corresponding data structures.

4.3.2. Raw Public Key Implementation

Of course, the use of raw public keys does not only impact the code size but also the size of the exchanged messages. When using raw public keys (instead of certificates) the "certificate" size was reduced from 475 bytes to 163 bytes (using an RSA-based public key). Note that the SubjectPublicKeyInfo block does not only contain the

raw keys, namely the public exponent and the modulus, but also a small ASN.1 header preamble.

For the raw public key implementation the following components were needed (in addition to a subset of the cryptographic support functions):

Minimal ASN1 Parser: 3,232 bytes

The necessary support from the ASN1 library now only contains functions for parsing of the ASN1 components of the SubjectPublicKeyInfo block.

Generic TLS Library: 16,288 bytes

This size of this library was slightly enlarged since additional functionality for loading keys into the bigint data structure was added. On the other hand, code was removed that relates to certificate processing and functions to retrieve certificate related data (e.g., to fetch the X509 distinguished name or the subject alternative name).

TLS Client Library: 4,528 bytes

The TLS client-specific code now contains additional code for the raw public key support, for example in the ClientHello message. Most functions were left unmodified.

5. Summary and Conclusions

TLS/DTLS can be tailored to fit the needs of a specific deployment environment. This customization property allows it to be tailored to many use cases including smart objects. The communication model and the security goals will, however, ultimately decide the resulting code size; this is not only true for TLS but for every security solution. More flexibility and more features will ultimately translate to a bigger footprint.

There are, however, cases where the security goals ask for a security solution other than TLS. With the wide range of embedded applications it is impractical to design for a single security architecture or even a single communication architecture.

6. Security Considerations

This document discusses various design aspects for reducing the footprint of (D)TLS implementations. As such, it is entirely about security.

7. IANA Considerations

This document does not contain actions for IANA.

8. Acknowledgements

The authors would like to thank the participants of the Smart Object Security workshop, March 2012. The authors greatly acknowledge the prototyping and implementation efforts by Pedro Moreno-Sanchez and Francisco Vidal-Meca who worked as interns at Philips Research.

9. References

9.1. Normative References

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[SHA] National Institute of Standards and Technology, , "Secure Hash Standard", FIPS 180-2, Aug 2002.

9.2. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC4785] Blumenthal, U. and P. Goel, "Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS)", RFC 4785, January 2007.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS)

Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC3268] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[RFC4101] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005.

[RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[TLS-IANA] IANA, "Transport Layer Security (TLS) Parameters: <http://www.iana.org/assignments/tls-parameters/tls-parameters.xml>", Oct 2012.

[I-D.ietf-tls-oob-pubkey] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress) February 2013.

[I-D.keoh-lwig-dtls-iot] Keoh, S., Kumar, S., and Garcia-Morchon, O., "Securing the IP-based Internet of Things with DTLS", draft-keoh-lwig-dtls-iot-01, February 2013.

[I-D.gilger-smart-object-security-workshop] Gilger, J., and Tschofenig, H., "Report from the 'Smart Object Security Workshop', March 23, 2012, Paris, France", draft-gilger-smart-object-security-workshop-01.txt, February 2013.

[Dunkels-Contiki] Dunkels, A., Gronvall, B., and Voigt, T. "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors", In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, IEEE, 2004.

[Bergmann-Tinydtls] Bergmann, O. "TinyDTLS - A Basic DTLS Server Template", <http://tinydtls.sourceforge.net>, 2012.

Authors' Addresses

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo, 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Sandeep S. Kumar
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: sandeep.kumar@philips.com

Sye Loong Keoh
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: sye.loong.keoh@philips.com