Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 6, 2012

J. Arkko
H. Rissanen
S. Loreto
Z. Turanyi
O. Novo
Ericsson

July 5, 2011

Implementing Tiny COAP Sensors
draft-arkko-core-sleepy-sensors-01

Abstract

The authors are developing COAP and IPv6-based sensor networks for environments where lightweight implementations, long battery lifetimes, and minimal management burden are important. The memo shows how different communication models supported by COAP affect implementation complexity and energy consumption, far more so than mere changes in message syntax. Our prototype implements a multicast-based IPv6, UDP, COAP, and XML protocol stack in less than 50 assembler instructions. While this extremely minimal implementation is suitable only for limited applications and makes a number of assumptions, the general conclusions point to need for further work in developing the COAP multicast and observation frameworks.

Table of Contents

1. Introduction

The authors are developing COAP [I-D.ietf-core-coap] and IPv6-based [RFC2460] sensor networks for home, building, and other consumer environments. These environments demand solutions where the sensors are physically small, inexpensive, have long battery lifetimes, and require minimal amount of management effort. Our prototype sensor implementation requires no configuration and implements a multicast-based IPv6, UDP, COAP, and XML protocol stack and an application in very small amount of code.

Small devices are naturally preferred in most applications, but for some applications small enough size is a critical concern, for instance, to make devices embedded in our clothing practical, to fit within the space available in buildings or everyday objects, or to ensure that the devices do not cause a visual distraction. Another key concern is device and battery lifetime. Sufficient battery lifetime in an application with a large number of devices can be surprisingly long. A home with hundred devices with ten year battery lifetimes will result in a battery change operation every month.

The practical challenge is to increase battery lifetimes of small devices by several orders of magnitude, and to enable pinhead size devices connected to the Internet. These are not unattainable goals, as legacy sensor networking technology can in some cases reach these goals. For instance, networked 1-Wire temperature sensors are the size of a packaged transistor. Our aim is to replicate this model or even improve it for IP-based sensors.

Another challenge is to ensure that COAP-based networks are interoperable in a multi-vendor environment. For instance, it is important that proxies and servers can perform all the necessary tasks without being programmed to support a sensor node manufactured by a particular vendor, or perhaps even without being programmed to support a particular class of a sensor.

This memo describes implementation experiences, open questions that we have encountered, and areas where COAP makes it difficult to make very low power implementations. The memo discusses implementation techniques that are useful in these environments and what is needed for fully interoperable solutions based on COAP.

The goals for our work are described in Section 2. Before we can dwell into the high-level networking design choices, we highlight some of the implications of detailed implementation strategies through an example. Section 3 discusses our specific implementation strategies, and describes our experiences with these choices. This example is an extreme case, an attempt to minimize as much as

possible for a limited set of applications. However, some general conclusions can still be seen. The more general discussion of the different high-level approaches to communications models can be found in Section 4. Different communication models supported by COAP affect implementation complexity and energy consumption, far more so than mere changes in message syntax. The required configuration effort is also directly affected by the choice of the communications model. Finally, the concluding recommendations point to need for further work in developing COAP and its multicast and observation frameworks [I-D.ietf-core-coap] [I-D.ietf-core-observe], as discussed in Section 6.

2. Goals

The main focus of this draft is sensors that are deployed in large quantities and have specific physical requirements. There are similar issues with other nodes such as servers and proxies, but in general these nodes have better access to power and other resources, and typically can also be more easily configured by humans.

As discussed in the introduction, for sensors the overall requirements revolve around minimizing physical size, cost, management effort and maximizing battery lifetimes. More specifically, we believe the following goals are key in achieving fulfilling these requirements:

- Natural support for sleeping nodes. There are many aspects to power usage in small devices, but we believe this one is the most significant one in terms of minimizing power usage. Many of the other aspects are either dictated by the environment (such as choice of radio technology in a given network) or have a relatively small impact (small variations in message size, for instance).

- Communication models that fit the problem at hand. It is essential that the small nodes can engage in communication exchanges that suit their needs. Having to employ multiple roundtrips, wait for nodes they have no control over, and so on can have a large negative effect on the amount of power that the node has to spend.

- Good design from user perspective. It is obviously undesirable to require a lot of per-device configuration effort when deploying a large set of small devices. In addition, direct configuration efforts with the device itself may be problematic, given that there is no room for any type of a user interface. For instance, some of the legacy sensor devices in existing networks are just a

few millimeters across. It is natural that some information needs to be configured, but configuration should be minimized and whatever configuration is necessary should take place in nodes that have the necessary user interfaces and capabilities.

3. Implementing Tiny COAP-Based Sensors

We have implemented prototypes of small sensors and a sensor gateway to pass the information onwards. The main target of these implementations is temperature, humidity, and other measurements in home environments. Our focus is primarily in sensing. Actuators and other more complex functions are outside the scope of our analysis.

Our prototype sensor implementation requires no configuration and simply runs based on its own identity burned in the hardware. The complete functionality requires only a small amount of code. Our prototype platform uses a 32-bit processor architecture and the hardware provides an underlying capability to send a link layer frame. In this platform, our implementation is under 50 assembler instructions (see Note 1). The implementation consists of a Ethernet, IPv6, UDP, COAP, and XML protocol stack and the sensor application. The sensor application is based on values provided by an A/D converter; any analog value can be measured.

Note 1: This contains everything necessary for using an A/D converter, constructing a complete Ethernet frame along with the higher level protocol fields, and asking the link layer to send it. It excludes platform specific initializations of the link layer, actual emitting of bits on the wire, and putting the device to sleep and waking it up. The complexity of these tasks varies highly between platforms and link layer technologies. For instance, one some platforms and with Ethernet, sending a frame out is merely a question of setting some registers and asking the hardware to send a packet. Of course, with different link layers and platforms an implementation might be have to be arbitrarily complex to support the intricacies of the link layer in question.

Even this size of the implementation is not the absolute minimum. One quarter of the code in our implementation relates to specific initializations required for the A/D converter that we used. Another quarter relates to binary to decimal conversions on the chosen XML-based payload. On a different platform and with binary data, 25 instructions would be achievable.

The following subsections outline the design choices that were taken to create the small implementation we have.

3.1. Sleeping Nodes and Energy Use

As discussed earlier, choosing the right communication model is what drives a good design from a power conservation perspective.

In our implementation, we chose to use a send-only model where the device only sends messages, but never receives one. This model can be applied in specialized applications under some assumptions that will be discussed further later in this memo. In our case, the sensor will periodically take a reading and send a COAP message to the network with that reading. In order to eliminate potential waiting periods where the device has to stay on, we needed to eliminate the following:

- DHCP request - response process [RFC2131].
- Router Discovery process [RFC4861].
- Duplicate Address Detection process [RFC4862].
- Acting as a COAP server [I-D.ietf-core-coap].
- Waiting for COAP observation subscriptions [I-D.ietf-core-observe].

3.2. Address Autoconfiguration

Eliminating DHCP is easy, as we can simply use IPv6 and stateless address autoconfiguration. Eliminating router discovery is harder, however. To avoid having to wait for a Router Advertisement to carry a prefix, we chose to employ a link-local source adress. These addresses can be constructed from the well-known prefix FE80::0 and a link layer hardware address burned to the hardware [RFC4862].

Eliminating Duplicate Address Detection is a matter of choice. We chose to behave as if DupAddrDetectTransmits had been set to zero, in other words not performing any Duplicate Address Detection. It may be debatable whether this is a violation of [RFC4862], but it is certainly against its spirit. This choice seems to be the right technical action, however, on a number of grounds:

- As the node is not receiving any packets, nor sending Neighbor Advertisement messages, any effects of possible duplication would be limited to some additional traffic in the network. No other traffic would be impacted. Application-level collection of sensor information can proceed even in this situation.

- o [RFC4862] requires that upon detecting a duplicated address, "autoconfiguration stops and manual configuration of the interface is required". However, it is obvious that no such action is possible on a small device. The device has no user interface. The only interface that the device has is the network, and if the network cannot be brought up, there's very little that can be done. As a result, the ability to not stop in a duplication case may actually be better than what is recommended by [RFC4862].

- o These devices are manufactured with hardware identities that are expected to be unique. There are obviously no guarantees about this succeeding in all cases, but non-unique identifiers would represent a major failure of the manufacturing process.

Elimination of Duplicate Address Detection also eliminates the need for the node to implement Multicast Listener Detection (MLD) protocol [RFC2710] [RFC3810]. This is because it now no longer needs to listen for messages to the solicited node multicast address, so there is no need to send out MLD messages.

3.3. Using Multicast

To further eliminate configuration or protocol exchanges for discovery, we chose to employ a multicast model where the sensor sends COAP POST requests to a well-known multicast address. While the type of sensors targeted here send information very infrequently, one of our goals was to ensure that the architecture would scale to more frequent information distribution and far larger groups of sensors. As a result, it was important to ensure that the multicast messages do not lead to multicast storms or unnecessary waking up many nodes due to frequent messages.

We chose to employ an interest-based generated multicast group address. These addresses are similar to those used in IPv6 Neighbor Discovery [RFC4861] for sending messages to solicited node addresses (FF02:0:0:0:0:1:FFXX:XXXX) [RFC4291]. The idea is that some bits from the object of interest are reflected in the multicast address, making it statistically likely that someone interested in a specific object only has to receive packets relating to that object, and not all packets.

We employ FF02:0:0:0:1:FEXX:XXXX, where XX:XXXX is a 6-byte value representing the type of sensor. (This address is currently reserved by IANA, but could be allocated for this purpose if needed.) The sensor type represents a classification of different sensor to types. For instance, we could let 00:0001 stand for temperature sensors. Each temperature sensor would send information to the multicast address FF02:0:0:0:1:FE00:0001, and only those devices that are

interested in temperature measurements would subscribe to this multicast group. Techniques such as MLD/IGMP snooping can be used in the network to ensure that multicast messages are physically transmitted only in those parts of the network that actually care about those messages [RFC4541]. In practice, this would mean that in a star topology network with a large number of sensors and a few central nodes, none of the sensors would receive any messages from each other.

Finally, randomization of actual transmission times for the periodic transmissions ensures that transmissions from different sensors are not synchronized.

When sensors send multicast messages with link-local source and destination addresses, all communication is confined to a single network. We expect that there is a node in the network that listens to the multicast messages, collects the data from them, and is capable of relaying the information to other parties. Such a node might store the latest information related to each sensor, and allow other nodes in the Internet to query the latest information on a per-sensor or an aggregate basis.

3.4. Using COAP

Our implementation uses non-confirmable requests at the messaging layer of COAP, and sends a POST message that carries an XML payload for a well-known URI. The implementation sends a message and does not wait for a message at this layer. We have used a gateway to store the information received from the sensors, making the gateway act as a server, storing everything posted to it. The stored information can be fetched from the gateway, for instance, with a COAP or HTTP GET.

Per Section 2.8.2 of [I-D.ietf-core-coap], POST methods normally generate a response at the request/response layer. If the server sends a response, the sensor is already asleep and will not respond to Neighbor Discovery messages or receive the actual message. The message is therefore lost, but it is fine in our case given that the information was already stored in the server.

Reliable transmission is achieved through assuming a sufficiently high periodic transmission rate to account for randomly occurring message loss.

There are several areas of concern with the above arrangements, discussed further in Section 3.8 and Section 4.2.

3.5. Power Usage Calculation

Our communication model is now complete. Its effectiveness can be calculated by determining what fraction of time the device would have to be awake. Lets assume periodic messages once per minute, a 10 Mbit/s link layer interface, and a CPU running at 1 Mhz. With the given link layer, sending one message takes theoretically 100 microseconds. Constructing the message takes 50 instructions and if we for simplicity assume that each instruction takes two clock cycles, the CPU needs to run for an additional 100 microseconds. Since our device is only sending messages, it only has to wake up to send the message. Ratio of sleeping versus being awake is now 200 microseconds versus 60 seconds, i.e., 300.000. Even if we assume that it takes an additional 800 microseconds to power the device up and let the A/D converter stabilize, the ratio is still 1000 microseconds vs. 60 seconds, i.e., 60.000.

We can compare this to some other possible implementations. A node that stays awake and participates in Neighbor Discovery, Duplicate Address Detection, and ARP processes would consume 60.000 times more energy. One could assume that listening is less power consuming than sending, however. On some link layers today this ratio can be as high as listening consuming 2.500 time less power, though practical implementations (talk vs. standby times) seem to be more in the range of a 100-fold difference. If we assume an optimistic 1.000 time difference, our implementation would still consume 60 times less energy than one that stays on all the time.

Another possible implementation is that a node stays awake for a short period of time to listen for possible messages. Some COAP implementations do this to enable discovery and observe subscriptions to work. If we assume one second awake time during one minute, then the power consumption difference to our implementation is somewhere between 1.000 and 2 times, depending on whether send/receive power requirements differences are factored in.

While these comparisons have produced wildly different numbers, it is clear that our implementation strategy is far superior to the simplistic always-on model. The situation is less clear with the comparison to the periodically listening approach, but even there it is clear that not listening consumes less energy than listening. While the actual numbers depend highly on the characteristics of the link layer, even with the most optimistic assumptions for the alternative approach it uses twice as much energy. This may not sound like a significant difference, but if it means a ten year battery lifetime instead of five year battery lifetime, it can make or break a business case for building some types of sensors.

3.6. Software Construction

When memory and processing power is at a premium, the detailed software design approach needs to suit the platform that the software runs on. That being said, for simple send-only applications we have found that a packet template-based approach works well. In this approach an image of the message sent by the application is burned into the read-only memory of the device, as a part of the overall software image. When the device powers up, the message image is copied to random access memory, necessary changes are applied, and the underlying link layer hardware or the CPU emits it on the outgoing interface bit by bit.

In case of COAP and simple sensors that output a numerical value transmitted in an XML [W3C.REC-xml-19980210] or JSON [RFC4627] payload, only the following changes are applicable:

- 16-bit COAP Message ID field (see Section 3.1 in [I-D.ietf-core-coap]). This field should be set to a random value, a rarely repeating value. We have found that using a suitably shifted value of a real-time clock is the most convenient way to generate a good value for this field. On many small platforms, a real-time clock can be kept counting with a very small amount of power. Note that it does not matter what value the real-time clock is initially initialized to; the only thing that matters for the Message ID field is that it keeps changing. If a sensor sends a value every minute, shifting a seconds-from-epoch counter by five bit positions is a good way to generate a unique value.

  Note that using a different value may not actually be required, though it is certainly helpful for understanding network traces and debugging. According to Section 4.1 of [I-D.ietf-core-coap], Message IDs only have to be unique within RESPONSE_TIMEOUT * RESPONSE_RANDOM_FACTOR * (2 ^ MAX_RETRANSMIT - 1) or 45 seconds, so a sensor sending messages every minute would be allowed to send them with the same Message ID.

- The actual sensor reading. In both XML and JSON, values can be padded with leading zeros or spaces, so the overall size of the packet can be kept the same in all circumstances. This greatly simplifies the construction of the packet.

  Note that binary or hexadecimal formats would make this even simpler, but the savings are in the order of few instructions; the difference is not big. Of course, a message that carries a text is longer than a pure binary message. However, the format is not so important as is avoiding including a lot of extraneous

information. Some XML schemas can be problematic. We advocate simplicity and restraint in XML schema design for sensor data.

- 16-bit UDP checksum field. For computing this field, see Section 3.7.

Note this small set of changes is only applicable when it can be assumed that both source and destination IP addresses are known beforehand.

3.7. UDP Checksums

Both IPv4 and IPv6 have some form of mandatory checksums, either in the IP header (IPv4) or as part of upper layer protocols such as UDP (IPv6). Computing the checksum is not difficult, but requires looping through all the 16-bit words in a packet. Fortunately, for a simple application the checksum calculation is actually very simple. Following the algorithm in [RFC1624], there is no need to calculate the checksum for the entire packet. The checksum can be precomputed on the packet template with zero words filled in for the variable parts. Lets call this precomputed checksum value C. Let NC be its negation, i.e.,

```
NC = ~C
```

Once the actual values are filled in the packet, the true checksum C' needs to be calculated as follows:

```
T = NC + W1 + W2 + ... + Wn
C' = ~(T + (T >> 16))
```

where T is a temporary variable and Wi, i = 0, 1, ..., n are the words that got changed from the template. Naturally, this approach makes sense only when the number of changed words is small. We have found that suitable placement of spaces and string values in an XML object, for instance, is helpful in aligning the changed parts to word boundaries, and in sensor implementations n = 3.

3.8. Evaluation

This type of an implementation is obviously an extreme example. This level of optimization may not be needed in all cases. Nevertheless, it is interesting to see that COAP can be used in such small implementations.

In general, our implementation satisfies the requirements set for the special environment that it was designed for: power usage is minimized, individual sensor devices do not require configuration,

existing legacy networks can migrate to general-purpose IP-based networks, and all the necessary information can be passed in the messages.

That being said, there are also some issues with this implementation approach. The first issue is that information delivery frequency is hardwired into the sensors. The chosen frequency may be sufficient for a given application, but the same sensors cannot be used by another application that would require a faster delivery of measurements.

Related but more serious concern is that reliability is achieved through randomized message intervals and multiple transmissions; it is considered unlikely that a very large number of messages in sequence are lost from the same sensor. The message transmission frequency needs to be set high enough to accommodate some packet loss. There is no way to actively request retransmission. We believe that this is a small problem in well-designed networks and for most applications that are not real-time critical, such as home, weather, maintenance, and energy monitoring. However, this approach may not be suitable for real-time or safety-critical applications.

The third and obvious limitation is that there is an assumption of a network node in the same network that is capable of storing information. We believe that there is little that can be done about this assumption; it is fundamental for the nature of low-power devices that they have to be able to sleep periodically, and there are very few other options beyond implementing a time-shifting device such as a cache. The location of the cache node could be outside the sensor network in some other designs, however.

4. Choosing a Communication Model

COAP is a specialized web transfer protocol designed to be used in various ways. The communication model of COAP is flexible and the application developer has to decide the best way to use it. This involves

- deciding which parties are in server/client roles,
- determining whether to use end-to-end communication or employ intermediary nodes,
- deciding whether to use base COAP operations or the observation framework,

- o deciding whether a discovery process is required,
- o specifying how COAP maps to lower layers, including choice of source and destination addresses, and
- o agreeing about commonly understood methods, resource identifiers and data representation.

Note that the number of these choices alone makes it hard to achieve interoperability, as we should strive for application interoperability at the semantic level [arkko.iab], rather than mere ability to transport correctly formed COAP packets.

Nevertheless, the main focus of this memo is to determine the power efficiency implications for the different communications models, and to identify areas where COAP limits this efficiency. The rest of this section is structured as follows. Section 4.1 discusses which nodes are involved. Section 4.2 discusses the specific COAP messaging alternatives. Section 4.3 discusses resources and data formats. Section 4.4 discusses configuration issues.

4.1. End-to-End Communication and Intermediaries

In most applications, user interactions and information requests can come at any time. Some form of an intermediary that can buffer such requests between a possibly sleeping device and the end user seems therefore useful to provide "time-shifting" capability. Similarly, an intermediary can be useful to reduce the number of transactions that one has to do with the low-power device to a minimum; the intermediary can answer on behalf of the device should a large number of information requests be placed.

In its simplest form, the intermediary is a part of the application server. For instance, a web-based application server is capable of serving web clients at any time, but will only place a periodic request to the sensor in order to take a reading. There are virtually no downsides to this arrangement, and it is generally recommended practice.

What is perhaps more controversial and interesting is the placement of intermediaries elsewhere, such as requiring an intermediary in the same network as the sensor devices are in. In our example implementation, such an intermediary was used for both time-shifting purposes and to bridge the gap between addressing domains, as the sensor was only capable of sending messages to nearby devices with link-local multicast addressing. For obvious reasons, sending traffic to well-known multicast groups works only on the local scale. Other possible reasons for using a local intermediary include

protocol conversion and providing TCP-based congestion control for traffic passing through the Internet. Where mechanisms for dealing with packet loss are limited, such as in the case of our implementation, an intermediary can also shield the sensors from having to deal with networks that have not been engineered for this purpose.

There are also downsides to having to place a local intermediary. The obvious downside is that such a device must now exist in the local network.

The use of COAP intermediaries is not fully specified, however. Some of the issues we have encountered include:

- COAP defines the roles for clients, servers, caches, and proxies, but while the specification allows an intermediary to act as server that stores all information sent to it, it is by no means specified as something that all implementations should do. The desirable behavior from the point of low-power sensors would be that the local server would store the information from every POST sent to it for a period of time specified in the Max-Age option [I-D.ietf-core-coap], and then be able provide access to the information using GET and HTTP/COAP. It would be useful to define such a new server role, along with specifying the necessary security and operational conditions for this practice.

- If designed badly, the intermediary may also limit the type of communications it can relay. For instance, a gateway that is only built for a particular types of sensors might only accept very specific COAP messages. In particular, intermediaries need to support any type of resource identifiers and data formats. Further discussion of this can be found in Section 4.3.

- In several CoAP applications the user is interested in the latest value of a resource, but historical values are also interesting in several use cases, e.g. tracking the movements of a truck during the day. Thus, the information stored in the cache/gateway should not expire. Even if a new value is received every minute, old values should be accessible and new value should not overwrite the old value. For this kind of cases, schemas for representing also historical values of the sensor would be useful for interoperability. Of course, simple schemas are easy to implement even if there did not exist any standards or recommendations, but again, there will not be interoperability.

- If the information is such that it should expire after some time, Max-Age option can be used as defined [I-D.ietf-core-coap]. However, [I-D.ietf-core-coap] discusses Max-Age option only in the

context of responses. In the multicast use case (sensor the one sending requests), Max-Age option would be needed to be supported in requests, too.

- o Multicast requests, particiularly GETs, might be forwarded by several proxies and possibly even to further multicast addresses, causing a storm of messages. The COAP specification does not describe when the forwarding of multicast requests is appropriate and when it is not.

4.2. COAP Messaging

The interaction model of COAP is similar to the client/server model in HTTP. A sensor can act either as a client that sends requests containing updated measurement information to a server, or as a server that responds to requests from others. If the sensor is a server, it can either employ the basic communication model from [I-D.ietf-core-coap] or use the observation framework [I-D.ietf-core-observe]. This section looks at the energy efficiency implications of these models.

It is important to make this analysis not merely based on the data transmission phase, but also based on what discovery actions and related signaling may be necessary.

4.2.1. Client Model

In this model, a sensor acts as a client that periodically sends POST requests containing updated measurement information to a server. This is the model that we used in our example implementation.

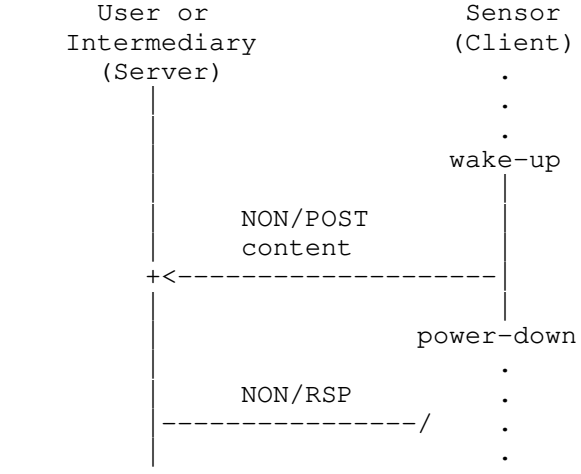


Figure 1. Send-only client model

In its simplest form, this model can be reduced to sending a single message per observation period, however this comes at the cost of:

- Limited support for reliable transmission. Messages may arrive out of order and they may go missing without notice. While periodic retransmissions do provide a statistical likelihood that the transmission eventually succeeds, they do not guarantee it.

- Possible spurious diagnostic or other problems caused by not being able to receive the REST level response to the POST message that the server will send (see Section 3.4).

Both of these problems can be addressed by forcing the device to wait for a response, incurring the cost of having to be awake for 1 RTT for each observation period. Using the assumptions from Section 3.5 and a 2 ms RTT for a local intermediary to respond, the power usage of this model would be either two times more or 0.2% more, depending, again, on whether the send/receive power differences are factored in.

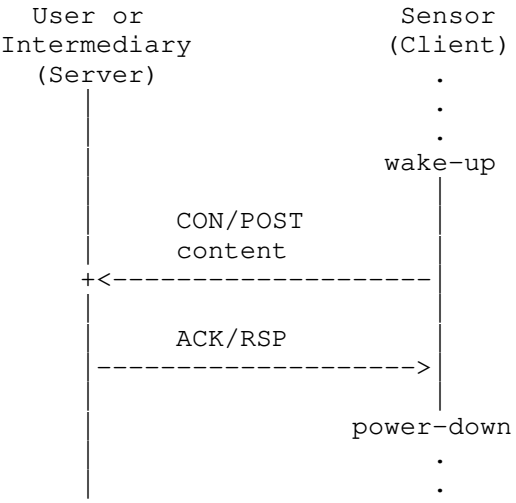


Figure 2. Send-and-confirm client model

(Interestingly, a similar model could be implemented even with HTTP. With TCP, one additional roundtrip and one additional message would be necessary to start the communications. This model would be roughly twice as power hungry as the COAP alternative. Note at least in the implementation strategy that was used in our example implementation, the format differences between COAP and HTTP would

make little difference for implementation complexity, as messages are created based on pre-filled packet templates. Supporting TCP would require some complexity, however.)

In addition, there is an added factor, having to discover the right peer to send messages to. In our example implementation this was simply a well-known multicast address, in which case no additional power is spent. The downside is that this can easily be done only with local multicast, necessitating the existence of suitable intermediary in the same network. Alternatively, the sensor could run a discovery phase at installation time to find the addresses of the peers wishing to receive the information. This discovery would have to repeated in order to account for changes and new equipment. Nevertheless, if discovery is run once a day and uses the same amount of power as sending one data observation, the increased power requirements are in the order of 0.1%, i.e., negligible.

The COAP specification has also a few more detailed issues around the use of the client model:

- Section 4.2 of [I-D.ietf-core-coap] indicates that multiple transmissions can be used to increase the reliability of non-confirmable requests. However, no rules are given about how many repetions can be made or how quickly they can follow each other. The specification also does not say if the rules are the same for unicast and multicast.

- Section 4.4 of [I-D.ietf-core-coap] does not explain the congestion control rules for multicast requests. There is an informative reference to another draft, but even that draft does not specify the behavior for multicast.

4.2.2. Server Model

In the basic server model as defined in [I-D.ietf-core-coap], the sensor waits for requests from a client. The power requirements for this model have been analyzed in Section 3.5 and are substantially higher than in any other model, even if one takes into account that listening is less power intensive than sending.

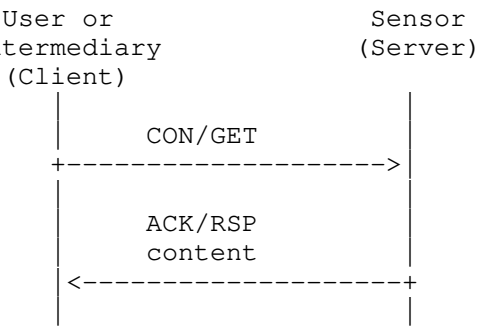


Figure 3. Server model.

There may be an additional discovery exchange where the sensor responds to requests sent for the well-known resources defined in [I-D.ietf-core-link-format]. However, these additional exchanges do not change power requirements significantly, as the sensor already has to be awake at all times. A more relevant concern is perhaps unwanted or accidental traffic to the sensor or one of the multicast addresses it belongs to (such as all-nodes [RFC4291]). Such traffic may have to be replied to or ICMP error messages may have to be sent, consuming additional energy.

The server model is not recommended. Variations of the model may be a little bit more efficient, however. For instance, a local server could send multiple requests in an effort to randomly hit a period when the sensor is powered up. However, such practices would still generate a lot of traffic in the network, which might not be desirable. For instance, if the network involves low-powered RPL routers [I-D.ietf-roll-rpl], extra traffic would be harmful.

4.2.3. Observer Model

The observer model [I-D.ietf-core-observe] allows clients to decide what information they want and servers to decide when to send that information. The model involves an initial registration, followed by the server sending periodic notifications. These notifications can be timed appropriately, so that the sensor only needs to wake up at suitable times.

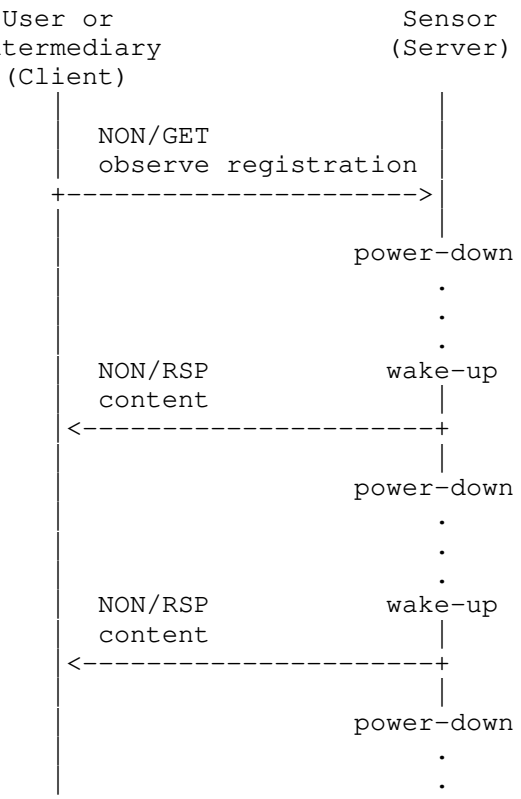


Figure 4. Observer model.

On the face of it, this is a very efficient model. Unfortunately, one has to take into account the registration phase. For this model to work, the sensor has to first be able to receive a registration request, and later be able to receive further requests in case there are changes or additional clients that want information. As a result, a straightforward implementation of the observe framework would appear to save no energy at all compared to the server model. The sensor would still have to stay awake all the time. Again, this model is not recommended.

Optimizations of the observer model are of course possible. Transmitting multiple registration requests is less damaging than transmitting multiple data requests, as the registration is only a one-time event. Nevertheless, for interoperability, it would be useful to understand what timelines and retransmission counts should be followed by both servers and clients. For instance, a sensor could assume that it has to be up one second out of every minute. This would increase power consumption compared to the send-only model

as described in Section 3.5. Users or intermediaries interested in subscribing to the information from the sensor would on the average have to re-transmit registration requests thirty times to randomly hit the period that a particular sensor is awake.

Another possible optimization would be the definition of implicit subscriptions where for some application a certain subscription would always be assumed so that a sensor can start sending periodic notifications immediately to a well-known address. With such a model the notifications are carried as responses and an intermediary can act as a COAP cache, avoiding most of the issues from the above paragraphs.

In addition, we have found a few more specific issues with the observer model:

- There is no well-defined termination period. The consumer of the information can observe that information is still flowing to it as expected. However, when non-confirmable messages are used, the sensor sending the notifications has no knowledge if the receiver is still even in the network. As a result, a simple implementation that keeps sending information until an explicit unsubscription is not desirable, as the sensor may have to send more messages than is necessary.

- Section 3.2 of [I-D.ietf-core-observe] specifies that a registration request from the same source address but a different port is considered a new, additional request. This can be problematic if the client reboots and assigns a different port number for its communication with the server.

- Section 3.3 of [I-D.ietf-core-observe] makes it optional for a server to terminate the observation request when a GET request is sent without the Observe option. This makes it hard for a client to indicate to the server that it is no longer interested in the resource.

- Section 3.3 of [I-D.ietf-core-observe] specifies that a subscription can be terminated using a RST message. This makes it impossible to know if the receiver rejects a confirmable notification because some context was missing or because the receiver wants to terminate the subscription.

- Section 3.3 of [I-D.ietf-core-observe] specifies that a timeout when sending a notification may be used to terminate a subscription. This seems like a drastic action for situations where it is important that the listener gets the information. For instance, using the observe model with a fire alarm would probably

not be a good idea if a temporary network problem could suddenly terminate a subscription.

- o A server may receive more subscription messages than it can handle. The base specification defines an error code (5.03 Service Unavailable) that could perhaps be used to reply in such situations, but it would be better if the behavior was explicitly specified and if it used a separate error code to make it clearer what the issue is.

4.3. Resources and Data Formats

The choice of resource identifiers (URIs) and data formats is important to achieve semantic interoperability between a sensor and an application using it. It is not enough to transport some data for some object, the parties involved in the application have to understand that the information comes from, say, a particular temperature sensor and that the information contains a temperature value encoded in a particular way.

The choice of URIs is clear as far as COAP transport is concerned in the server model. Here the Link Format [I-D.ietf-core-link-format] can be used by clients to find out what URIs exist. Nevertheless, there are two remaining concerns:

- o The authors of this memo found it desirable to implement a new URI type to represent device identities, such as MAC addresses or 1-wire device identifiers. While UUIDs [RFC4122] can also be used for this purpose, they are more complex for no additional value from the point of view of our application. UUIDs are required to contain a time component, which would cause both additional implementation complexity, as well as make it more difficult to correlate identifiers from a manufacturer's list or printed on the outside of the sensor to the ones actually sent in the network. (Such correlation is often required in order to configure the real-world location of various sensors.). The new URI type is simply of the form "device:ID", where ID is the hardware address associated with the device. Such an URI could have uses not only in sensor networks, but also in cataloging network equipment, etc.

- o While the Link Format provides a way to determine what resources exist, the semantics of those resources and data formats still require standardization. Some work regarding such standardization is ongoing, e.g. in ZigBee IP Smart Energy 2.0 Profile, but it remains to be seen how much work is needed overall. This problem might become even more real when sensors from particular application areas, such as electrical cars or lightning, are being implemented. Without any common schemas or data models no

interoperability can be provided.

- It is also important to care about the size and complexity of the data models developed for low-power applications. Even if moving from HTTP to COAP and some form of compression saves some number of bytes, complicated XML models can easily consume the savings and more. The authors have found [I-D.jennings-senml] a workable, simple model.

In addition, in the client model it becomes important that the server (local intermediary) is capable of storing information about any resource when it receives a POST request. This is not necessarily the case. First, it is unclear what resource identifiers the client should use, particularly when multicast is used. Our example implementation employed a well-known URI "/publish" and placed the identity of the device sending the request inside the payload part of the request along with the sensor readings. But it is not clear that this is the best approach, and furthermore, such an approach has not been standardized so it may not work with all devices. As an example, in one of the COAP stacks that we tried, it is only possible to generate resources by a user under a root resource called "storage". This requirement makes it incompatible with other implementations we tried.

4.4. Configuration

One overriding concern in networks with large number of sensors is configuration effort. In addition, the sensors are typically deployed in homes and other environments where the necessary skills for installation and operational tasks cannot be assumed. As a result, it is important that at installation of individual sensors leads to little or no configuration effort. Furthermore, given the small physical size and lack of user interfaces, it is essential that any configuration be doable on other devices on behalf of the sensors.

A good model for configuration is that the sensors are fully factory-configured with respect to their identities and capable of operating autonomously in any IP network with suitable network interfaces. Typically, some configuration information is required but this can be provided as additional information associated with a particular sensor identity, and configured in the application server or intermediary. For instance, the physical location of a sensor can be configured in this manner.

From the point of view of the COAP protocol and its communication model, this means that the sensors should operate as much as possible based on autoconfigured addresses, well-known destinations and/or

resource discovery [I-D.ietf-core-link-format] [I-D.shelby-core-resource-directory]. COAP should also allow configuration and passing of additional information in intermediaries.

5. Security Considerations

Support for authentication of sensors, integrity of messages sent by sensors, or protection of the data objects carried by the messages would be useful in some environments, while physical security and link-layer protection may be sufficient in others. Mechanisms for these security mechanisms are for further study.

6. Conclusions

This memo has analyzed the power requirements for sensor applications through an example implementation that runs on absolute minimum power and through an analysis of various different more general communications models.

The general conclusion is that the chosen communications model and overall system and network architecture is far more important for low power usage than details of the message formats. Much of the work in COAP has focused on the latter rather than the former. Even the difference between COAP and HTTP transactions is small compared to the difference between choosing the optimal and worst communications model.

In particular, we would like to draw attention to system-level analysis to ensure that nodes can stay asleep for as long as necessary. This is particularly important when designing power-efficient data transmission models such as the observe framework. It is not enough for the data transmission itself to be efficient if the device needs to stay awake or communicate for other reasons (Section 4.2.3). Several other more detailed observations about the COAP specifications were also noted in Section 3.3, Section 4.1, Section 4.2, Section 4.3, and Section 4.4.

The communication model is also not just about finding the most efficient sequence of messages. It is very much also an architectural decision. The authors believe that an information-centric or delay-tolerant networking model is appropriate for collecting information from sensor networks. These models allow communications based on identities, support intermittent connectivity, focus on data rather than the location of the data, and have the natural ability for nodes to aggregate, store, and process

data. Some of the tasks for ensuring that such models can be employed with COAP include

- Definition of URI types suitable to be used in sensor networks.
- Accurate specification of multicast support.
- Specifications for intermediary behavior so that they can store and process data from sensors.
- Further standardization of data formats and application semantics.

Finally, it should be noted that the conclusions in this memo should not be interpreted to apply too widely. Actuators and other, non-sensor low-power device implementations have likely very different requirements and may require different solutions.

7. References

7.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06 (work in progress), May 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-05 (work in progress), May 2011.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-02 (work in progress), March 2011.

[RFC1624] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

7.2. Informative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.

[RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4541] Christensen, M., Kimball, K., and F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4541, May 2006.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[W3C.REC-xml-19980210] Sperberg-McQueen, C., Bray, T., and J. Paoli, "XML 1.0 Recommendation", World Wide Web Consortium FirstEdition REC-xml-19980210, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210>.

[I-D.shelby-core-resource-directory] Shelby, Z. and S. Krco, "CoRE Resource Directory", draft-shelby-core-resource-directory-00 (work in progress), June 2011.

[I-D.ietf-roll-rpl] Winter, T., Thubert, P., Brandt, A., Clausen, T., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., and J. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks", draft-ietf-roll-rpl-19 (work in progress), March 2011.

[I-D.jennings-senml]
Jennings, C., "Media Type for Sensor Markup Language (SENML)", draft-jennings-senml-05 (work in progress), March 2011.

[arkko.iab]
Arkko, J., "Interoperability Concerns in the Internet of Things", Position paper at the IAB workshop on Smart Objects , March 2011, <http://www.arkko.com/publications/IAB-IOT-WS-Interoperability.pdf>.

Appendix A. Acknowledgments

The authors would like to thank to Magnus Westerlund, Ari Keranen, Stig Venaas, Zach Shelby, Cullen Jennings, Vlasios Tsiatsis, Jan Holler, Anders Eriksson, and Joel Halpern for their help and for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Heidi-Maria Rissanen
Ericsson
Jorvas 02420
Finland

Email: heidi-maria.rissanen@ericsson.com

Salvatore Loreto
Ericsson
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Zoltan Turanyi
Ericsson
Irinyi Jozsef u. 4-20.
Budabest
Hungary

Email: zoltan.turanyi@ericsson.com

Oscar Novo
Ericsson
Jorvas 02420
Finland

Email: oscar.novo@ericsson.com

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2012

C. Bormann, Ed.
Universitaet Bremen TZI

July 4, 2011

Guidance for Light-Weight Implementations of the Internet Protocol Suite
draft-bormann-lwig-guidance-00

Abstract

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

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

Status of this Memo

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

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

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

This Internet-Draft will expire on January 5, 2012.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

1.1. Objectives

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

Building a small implementation does not have to be hard. Many small

devices use stripped down versions of general purpose operating systems and their TCP/IP stacks. However, there are implementations that go even further in minimization and can exist in as few as a couple of kilobytes of code, as on some devices this level of optimization is necessary. Technical and cost considerations may limit the computing power, battery capacity, available memory, or communications bandwidth that can be provided. To overcome these limitations the implementers have to employ the right hardware and software mechanisms. For instance, certain types of memory management or even fixed memory allocation may be required. It is also useful to understand what is necessary from the point of view of the communications protocols and the application employing them. For instance, a device that only acts as a client or only requires one connection can simplify its TCP implementation considerably.

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

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

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

1.2. Call for contributions

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

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

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. As this is an informational document, the [RFC2119] keywords will only be used to underscore requirements where similar key words apply in the context of the specifications the light-weight implementation of which is being discussed.

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

2. Drawing the Landscape

There is not a single kind of constrained, Internet-connected device. To the contrary, the trend is towards much more functional variety of such devices than is customary today in the Internet. This section introduces a number of terms that will be used to locate some of the technique described in the following sections within certain areas of applications.

2.1. Classes of Devices

Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may, be worthwhile to have some succinct terminology for different classes of constrained devices. In this document, the following class designations may be used as rough indications of device capabilities:

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 1	~ 10 KiB	~ 100 KiB
Class 2	~ 50 KiB	~ 250 KiB

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

2.2. Design Objectives

- Consideration for design or implementation approaches for implementation of IP stacks for constrained devices will be impacted by the RAM usage for these designs. Here the consideration is what is the best approach to minimize overhead.

- In addition, the impact on throughput in terms of IP protocol implementation must take into consideration the methods that minimize overhead but balance performance requirements for the light-weight constrained devices.

- Protocol implementation must consider its impact on CPU utilization. Here guidance will be provided on how to minimize

tasks that require additional CPU execution time.

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

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

2.3. Implementation Styles

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

...

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

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

2.4. Roles of nodes

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

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

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

- Bandwidth/latency considerations

2.5. Overview over the document

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

3. Data Plane Protocols

3.1. Link Adaptation Layer

6LoWPAN

3.1.1. Fragmentation in a 6LoWPAN Route-Over Configuration

Author: Carsten Bormann

6LoWPAN [RFC4944] is an adaptation layer that maps IPv6 with its minimum MTU of 1280 bytes to IEEE 802.15.4, which has a physical layer MTU of only 127 bytes (some of which are taken by MAC layer and adaptation layer headers). Therefore, the adaptation layer provides a fragmentation and reassembly scheme that can fragment a single IPv6 packet of up to 1280 bytes into multiple adaptation layer fragments of up to 127 bytes each (including MAC and adaptation layer overhead).

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

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

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

overflow data may need to be stored in the runt buffers to be combined with further fragments or may simply be forwarded as a separate additional fragment.)

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

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

3.1.1.1. Implementation Considerations for Not-So-Constrained Nodes

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

3.2. Network Layer

IPv4 and IPv6

3.3. Transport Layer

TCP and UDP

3.4. Application Layer

3.4.1. General considerations about Application Programming Interfaces (APIs)

Author: Carl Williams

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

When an API implements a protocol, this can be based on proxy methods for remote invocations that underneath rely on the communication

protocol. One of the roles of the API can be exactly to hide the detail of the transport protocol.

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

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

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

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

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

3.4.2. Constrained Application Protocol (CoAP)

Author: Olaf Bergmann

The Constrained Application Protocol [I-D.ietf-core-coap] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios therefore include building automation and the Internet of Things. The major design objectives have been set on small protocol overhead, robustness against packet loss, and high latency induced by small bandwidth shares or slow request processing in end nodes. To leverage integration of constrained nodes with the world-wide Internet, the protocol design was led by the architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [RFC2616].

Lightweight implementations benefit from this design in many respects: First, the use of Uniform Resource Identifiers (URIs) for naming resources and the transparent forwarding of their representations in a server-stateless request/response protocol make protocol-translation to HTTP a straightforward task. Second, the set

of protocol elements that are inevitable for the core protocol and thus must be implemented on every node has been kept very small to avoid unnecessary accumulation of optional features. Options that -- 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 trade-off between robustness and latency of constrained nodes on one hand and resource demands (such as battery consumption, dynamic memory needs and static code-size) on the other. This section gives some guidance on possible strategies to solve this trade-off for very constrained nodes (Class 1 in Section 2.1). The main focus is on servers as this is deemed the predominant case where CoAP applications are faced with tight resource constraints.

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

3.4.2.1. Message Layer Processing

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

Where multi-threading is not an option at all because no full-fledged operating system is present, all operations are triggered by a big main loop in a send-receive-dispatch cycle. To implement the packet retransmission, CoAP implementations at least need a separate send buffer and a decent notion of time, e.g. as a strictly monotonic increasing tick counter. For platforms that disable clock tick interrupts in sleep states, the application must take into consideration the clock deviation that occurs during sleep (or ensure to remain in idle state until the message has been acknowledged or the maximum number of retransmissions is reached). Since CoAP allows up to four retransmissions with a binary exponential back-off it could take up to 45 seconds until the send operation is complete. Even in idle state, this means substantial energy consumption for low-power nodes. Implementers therefore might choose a two-step strategy: First, do one or two retransmissions and then, in the later

phases of back-off, go to sleep until the next retransmission is due. In the meantime, the node could check for new messages including the acknowledgement for any confirmable message to send.

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

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

3.4.2.2. Message Parsing

Both CoAP clients and servers must construct outgoing CoAP PDUs and parse incoming messages. The basic message header consists of only four octets and thus can be mapped easily to an internal data structure, considering the actual byte order of the host. Once the message is accepted for further processing, the set of options contained in the received message must be decoded to check for unknown critical options. To avoid multiple passes through the option list, the option parser might maintain a bit-vector where each bit represents an option number that is present in the received request. The delta-encoded option number indicates the number of left-shift operations to apply on a bit mask to set the corresponding bit.

In addition, the byte index of every option is added to a sparse list (e.g. a one-dimensional array) for fast retrieval. This particularly enables efficient reduced-function handling of options that might occur more than once such as Uri-Path. In this implementation strategy, the delta is zero for any subsequent path segment, hence

the stored byte index for option 9 (Uri-Path) will be overwritten to hold a pointer to the last occurrence of that option, i.e., only the last path component actually matters. (Of course, this requires choosing resource names where the combination of (final Uri-Path component, final Uri-Query component) is server-wide unique.

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

Once the option list has been processed at least up to the highest option number that is supported by the application, any known critical option and all elective options can be masked out to determine if any unknown critical option was present. If this is the case, this information can be used to create a 4.02 response accordingly. (Note that the remaining options also must be processed to add further critical options included in the original request.)

3.4.3. (Other Application Protocols...)

4. Control Plane Protocols

4.1. Link Layer Support

ARP, ND; 6LoWPAN-ND

4.2. Network Layer

ICMP, ICMPv6, IGMP/MLD

4.3. Routing

RPL, AODV/DYMO, OLSRv2

4.4. Host Configuration and Lookup Services

DNS, DHCPv4, DHCPv6

5. Security protocols

5.1. Cryptography for Constrained Devices

5.2. Transport Layer Security

TLS, DTLS, ciphersuites, certificates

5.3. Network Layer Security

IPsec, IKEv2, transforms

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

5.4. Network Access Control

(PANA, EAP, EAP methods)

5.4.1. PANA

Author: Mitsuru Kanda

PANA [RFC5191] provides network access authentication between clients and access networks. The PANA protocol runs between a PANA Client (PaC) and a PANA Authentication Agent (PAA). PANA carries UDP encapsulated EAP [RFC3748] and includes various operational options. From the point of view of minimal implementation, some of these are not necessary for constrained devices. This section describes a minimal PANA implementation for these devices.

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

5.4.1.1. PANA AVPs

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

Integrity-Algorithm AVP and PRF-Algorithm AVP:

All PANA implementations MUST support AUTH_HMAC_SHA1_160 for PANA message integrity protection and PRF_HMAC_SHA1 for pseudo-random

function (PRF) specified in [RFC5191]. Both of these are based on SHA-1, which therefore needs to be implemented in a minimal implementation.

Nonce AVP:

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

5.4.1.2. PANA Phases

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

Authentication and authorization phase:

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

EAP messages can be carried in PANA-Auth-Request and PANA-Auth-Answer messages. In order to reduce the number of messages, "Piggybacking EAP" is useful. Both the PaC and PAA should include EAP-Payload AVP in each of PANA-Auth-Request and PANA-Auth-Answer messages as much as possible. Figure 1 shows an example "Piggybacking EAP" sequence of the Authentication and authorization phase.

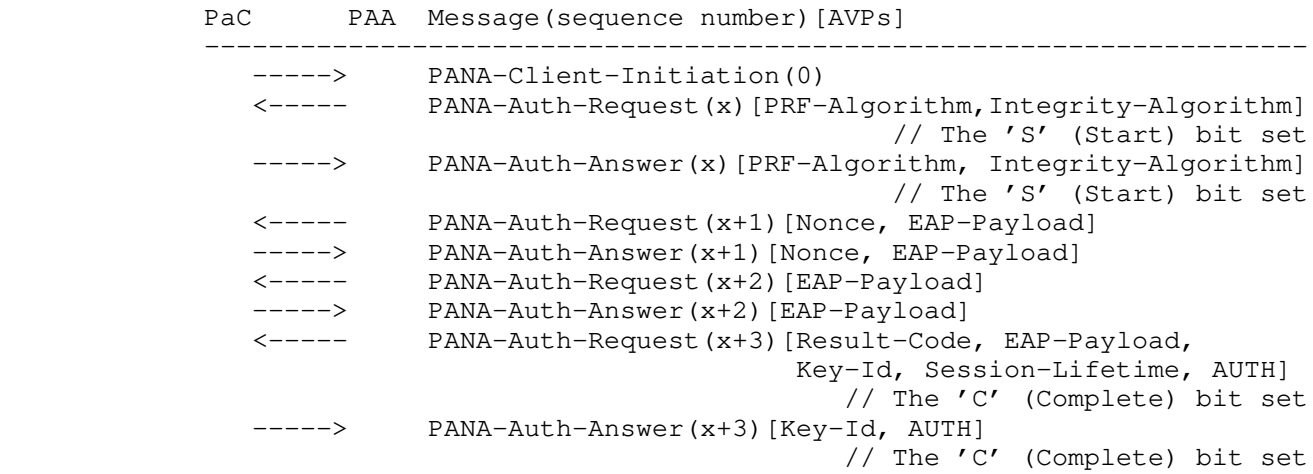


Figure 1: Example sequence of the Authentication and authorization phase for a PaC-initiated session (using "Piggybacking EAP")

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

Access phase:

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

Re-Authentication phase:

Before PANA session lifetime expiration, the PaC and PAA MUST re-

negotiate to keep the PANA session. This means that the PaC and PAA enter Re-Authentication phase. Also in the Authentication and authorization phase, there are two types of re-authentication. The minimal implementation must support PaC-initiated re-authentication and does not need to support PAA-initiated re-authentication (again because the PaC may be a sleeping device). "Piggybacking EAP" is also useful here and should be used as well. Figure 2 shows an example "Piggybacking EAP" sequence of the Re-Authentication phase.

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

Figure 2: Example sequence of the Re-Authentication phase for a PaC-initiated session (using "Piggybacking EAP")

Termination Phase:

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

5.4.1.3. PANA session state parameters

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

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

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

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

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

6. Wire-Visible Constraints

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

7. Wire-Invisible Constraints

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

8. IANA Considerations

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

9. Security Considerations

(TBD.)

10. Acknowledgements

Much of the text of the introduction is taken from the charter of the LWIG working group and the invitation to the IAB workshop on Interconnecting Smart Objects with the Internet. Thanks to the numerous contributors.

10.1. Contributors

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

Name	Affiliation
Carl Williams	MCSR Labs
Carsten Bormann	Universitaet Bremen TZI
Mitsuru Kanda	Toshiba
Olaf Bergmann	Universitaet Bremen TZI
...	...

11. References

11.1. Normative References

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

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

11.2. Informative References

[I-D.arkko-core-sleepy-sensors] Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-00 (work in progress), July 2011.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06 (work in progress), May 2011.

[I-D.kivinen-ipsecme-ikev2-minimal] Kivinen, T., "Minimal IKEv2", draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress), February 2011.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

[WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", ISBN 9780470747995, 2009.

Author's Address

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

Phone: +49-421-218-63921
Fax: +49-421-218-7000
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

A. Castellani
University of Padova
S. Loreto
Ericsson
March 14, 2011

Best Practice to map HTTP to COAP and viceversa
draft-castellani-core-http-coap-mapping-01.txt

Abstract

This draft aims at being a simple guide to the use of CoAP REST interface, to show how it can be mapped to and from HTTP, and at being a base reference documentation for CoAP/HTTP proxy implementors.

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 September 15, 2011.

Table of Contents

1. Introduction

Since implementing on constrained devices the full HyperText Transfer Protocol (HTTP) [RFC2616] is believed to be operationally and computationally too complex, especially in an M2M communication environment, resources available on constrained nodes are expected to be served using CoAP [I-D.ietf-core-coap].

> "The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles (called an end-point). A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation." Section 2 [I-D.ietf-core-coap]

These days the information is increasingly converging on the Web, thus an easy CoAP interoperability with HTTP is a paramount feature for CoAP. Indeed leveraging on both the easy CoAP/HTTP translation and the common usage of URI(s) to identify resources, it will become extremely simple to integrate constrained nodes in the Web.

The internetworking described in this document between CoAP and HTTP is mainly based on three points:

- the URI does not change between CoAP and HTTP, the scheme identifies the protocol;
- HTTP/CoAP mapping is performed by a proxy, both HTTP/CoAP endpoints can be not aware that a mapping is happening;
- using a named URI authority and DNS can be useful for the mapping.

The proxy itself does not require any particular knowledge about the constrained network topology, devices contained, nor about the content of data exchanged.

2. HTTP-CoAP

HTTP-CoAP mapping spans across several protocol layers:
- HTTP is mapped to CoAP
- TCP is used on the HTTP side, while CoAP uses UDP transport

In addition to this 6LoWPAN adaptation layer addresses a similar networking scenario, thus a convertion between IPv4/IPv6 to 6LoWPAN MAY be present as well.

2.1. URI

Any resource available in CoAP can be accessed using HTTP at the same URI, except for the scheme. The scheme represents the protocol used by the endpoint to access the resource.

The CoAP resource "//node.coap.something.net/foo" can be accessed using CoAP at the URI "coap://node.coap.something.net/foo", and using HTTP at the URI "http://node.coap.something.net/foo". When the resource is accessed using HTTP, the mapping from HTTP to CoAP is performed by a proxy

The usage of the same URI to access a resource, independently if it is accessed by a CoAP client within the same constrained network or by a HTTP client outside the constrained network, reduces the complexity of a proxy performing the mapping.

OPEN ISSUE: discuss the DNS usage resolving the URI.

2.2. Proxy

A device providing cross-protocol HTTP-CoAP mapping is called HTTP-CoAP cross-protocol proxy (HC proxy).

Usually regular HTTP proxies are same-protocol proxies, because can map from HTTP to HTTP. CoAP same-protocol proxies are intermediaries for CoAP to CoAP exchanges, however the discussion about that entities is out-of-scope of this document.

At least two different kinds of HC proxies may exist:

- One-way cross-protocol proxy (1-way proxy): It can translate from a client of a protocol to a server of another protocol but not viceversa.
- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): It can translate from a client of both protocols to a server of the other protocol.

1-way and 2-way HC proxies can be realized using the following general types of proxies:

Forward proxy (F): It is a proxy known by the client (either CoAP or HTTP) used to access a specific cross-protocol server (respectively HTTP or CoAP). Main feature: server(s) do not require to be known in advance by the proxy (ZSC: zero server configuration).

Reverse proxy (R): It is known by the client to be the server, however for a subset of resources it works as a proxy, by knowing the real server(s) serving each resource. When a cross-protocol resource is accessed by a client, the request will be silently forwarded by the reverse proxy to the real server (running a different protocol). If a response is received by the reverse proxy, it will be mapped, if possible, to the original protocol and sent back to the client. Main feature: client(s) do not require to be known in advance by the proxy (ZCC: zero client configuration).

Transparent (or Intercepting) proxy (I): This proxy can intercept any origin protocol request (HTTP or CoAP) and map it the destination protocol, without any kind of knowledge about the client or server involved in the exchange. Main feature: client(s) and server(s) do not require to be known in advance by the proxy (ZCC and ZSC).

The proxy can be placed in the network at three different logical locations:

Server-side proxy (SS): a proxy placed on the same network domain of the server;

Client-side proxy (CS): a proxy placed on the same network domain of the client;

External proxy (E): a proxy placed in a network domain external to both endpoints.

In the most common scenario the HC proxy is expected to be server-side and deployed at the edge of the constrained network. The arguments supporting this assumption are the following:

TCP/UDP: Translation between HTTP and CoAP requires also a TCP to UDP mapping; UDP performance over the Internet may not be adequate, UDP should be dropped as soon as possible to minimize the number of required retransmissions and overall reliability.

Multicast: To enable access to local-multicast in the constrained network, the HC proxy may require a network interface directly attached to the constrained network.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, network edge is a strategical placement for this need.

Security: HTTPS sessions should be terminated as near as possible to the CoAP server.

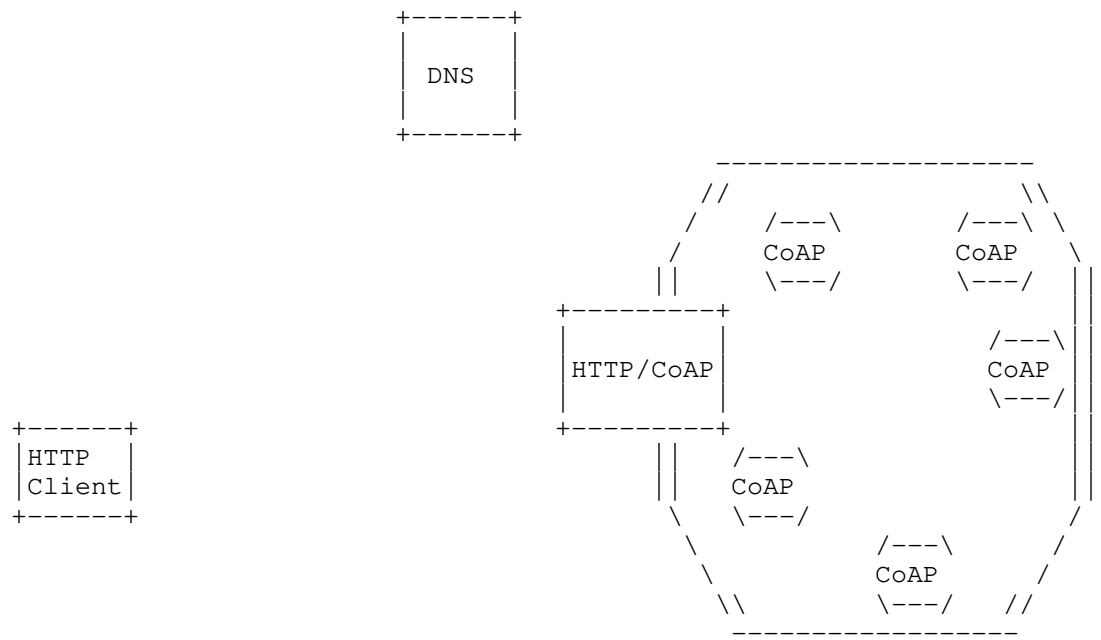


Table 1 shows some interesting HC proxy scenarios, and quickly marks the advantages related to each scenario.

Feature	F CS	R SS	I SS
TCP/UDP	-	+	+
Multicast	-	+	+
Caching	-	+	+
Security	?	+	-
Scalability	+	?	+
Configuration	-	-	+

Table 1: Interesting HC proxy deployments

The following open questions are left open in Table 1:
1. Are CoAP security modes adequate for Internet-wide operation?
2. Are reverse proxy setups scalable?

2.2.1. HC proxy discovery using DNS-SD

DNS-SD can be used by an HTTP client to discover the HC proxy in authority for a specific domain [I-D.jennings-http-srv].

An HTTP client wants access a resource that it knows being identified by the following URI:

//node.coap.something.net/foo

To find the address of the HC proxy, the HTTP client will look up the following SRV record:

_http._tcp.node.coap.something.net

The DNS will contain the following record:

```
      _http._tcp.node.coap.something.net  IN  SRV      0  1  80  hc-proxy.somet
hing.net
        hc-proxy.something.net  IN  A  192.168.0.1  ;  the address of the HC pr
oxy
```

The client will pass the request to the HC proxy that will translate it in a CoAP request. The CoAP side of the proxy will lookup the DNS in order to find the actual constrained device in authority for that URI.

2.3. Mapping

CoAP offers a subset of HTTP features in terms of methods, statuses and options supported; thus some HTTP request MAY NOT be mappable to CoAP.

In particular CoAP lacks the following methods defined in HTTP: OPTIONS, HEAD, TRACE and CONNECT.

An HC proxy receiving an HTTP request with a method not supported in CoAP MUST immediately drop handling the request and MUST send a response with status "405 Method Not Allowed" to the HTTP client.

The mapping of a CoAP response code to HTTP is not straightforward, this mapping MUST be operated accordingly to Table 4 of [I-D.ietf-core-coap].

The mapping of conditional HTTP requests is defined in Section 8.2 of [I-D.ietf-core-coap].

An HC proxy MUST always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority section of the URI is thus used internally by the HC proxy and SHOULD not be mapped to CoAP.

If an empty CoAP ACK is received, the actual CoAP response is

deferred. As described in CoAP specification the ACK is transparent to the HTTP client.

No upper bound is defined for a server to provide that response, thus for long delays the HTTP client or any other proxy in between MAY timeout, further considerations are available in Section 7.1.4 of [I-D.ietf-httpbis-p1-messaging].

If the HTTP client times out and drops the HTTP session to the proxy (closing the TCP connection), the HC proxy SHOULD wait for the response and cache it if possible. Further idempotent requests to the same resource can use the result present in cache, or if a response has still to come requests will wait on the open CoAP session.

Safe or non-idempotent requests MAY timeout. How the HC proxy should handle this situation?

The HC proxy MUST define an internal timeout for each CoAP request pending, because the CoAP server MAY silently die before completing the request. This timeout SHOULD be as high as possible.

Figure 2 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P).
node.coap.something.net has an A record containing the IPv4 address of the HC proxy, and an AAAA record containing the IPv6 of the CoAP server.

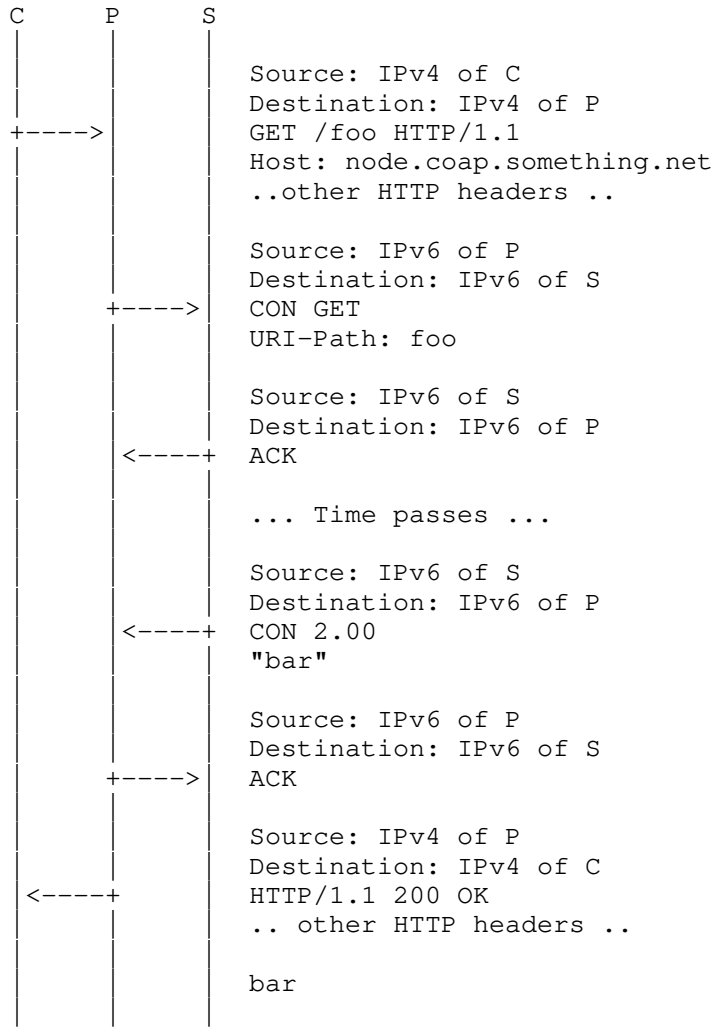


Figure 2: HTTP/IPv4 to CoAP/IPv6 mapping

The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. Thus IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typically expected use case.

When P is an intercepting HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

When the HTTP client has native IPv6 support, a convenient deployment choice should be to use an HC intecepting proxy. Thus the proxy MUST be located in the IPv6 network path between the client and the server, thus near to the server itself in order to support any Internet client.

2.4. Multiplexing CoAP responses

Defining the mapping of some advanced CoAP features to HTTP (i.e. multicast, observe) must address the need to asynchronously deliver multiple responses to the same HTTP request.

Some HTTP features are useful to succesfully represent these particular sessions.

Using Multipart media type is a suitable solution to deliver multiple CoAP responses within a single HTTP response.

Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.

An HC proxy may prefer to transfer each CoAP response immediately after its reception. Responses can be immediately transferred in "chunks" of an HTTP chunked Transfer-Encoding session, without knowing in advance the total number of responses and with arbitrary delay between them.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [I-D.loreto-http-bidirectional]. Large delays between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

When responses are coming from different sources, i.e. multicast, details about the actual source of each CoAP response SHOULD be provided. Source information can be represented in HTTP using a Link option described in [RFC5988] using "via" relation type.

Figure 3 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P). Discussion related to group communication in CoAP can be found in [I-D.rahman-core-groupcomm].

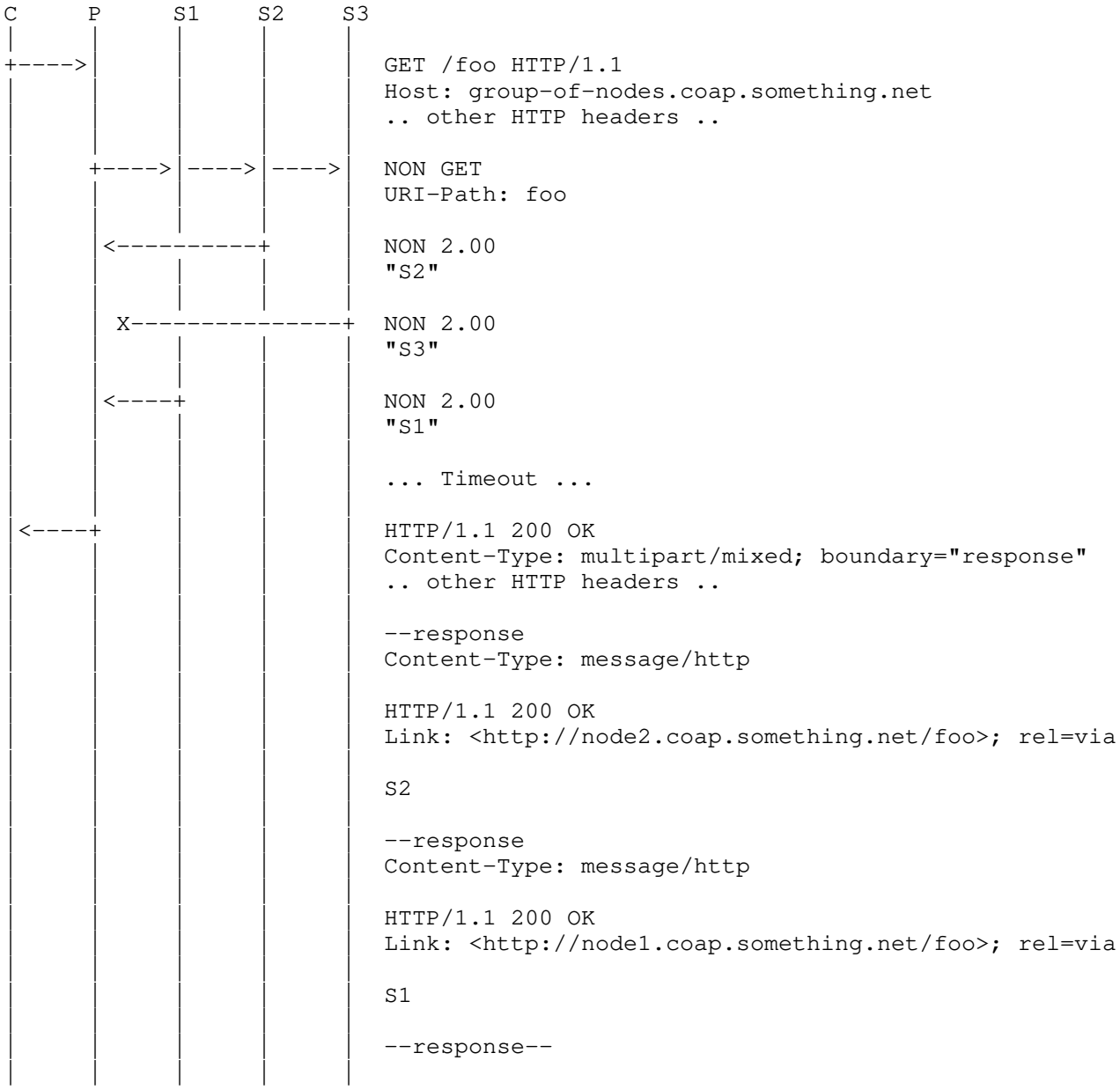


Figure 3: Unicast HTTP to multicast CoAP mapping

The mapping proposed in the above diagram does not make any assumption in how multicasting is done on the constrained network.

If IPv6 multicast support is present in the constrained network, an AAAA record containing the IPv6 multicast group will start multicast operation at the proxy. Otherwise the authority part of the URI is used by the HC proxy to match with a locally defined group of nodes.

In order to minimize the delay in delivering the responses (e.g. HTTP client can incrementally process the responses, HC proxy can reduce internal buffering), each CoAP response can be immediately streamed using HTTP chunked Transfer-Encoding. This encoding was not shown in order to simplify Figure 3, an example showing immediate delivery of CoAP responses is provided in Figure 4 (observe session).

2.4.1. Establishing a CoAP subscription

Using an exchange similar to the one shown in Figure 3, a CoAP observe session can be directly established by a willing HTTP client. Observe mechanism is specified in [I-D.ietf-core-observe].

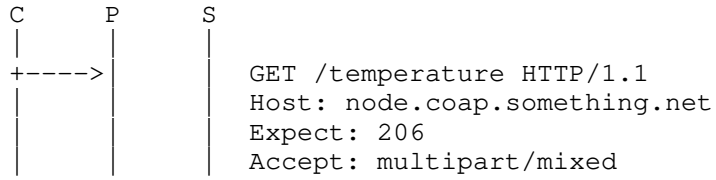
An HTTP client willing to establish a subscription to the "/temperature" resource of a CoAP server SHOULD send an HTTP request with Expect header set to "206" and Accept header set to "multipart/mixed".

The Lifetime of the subscription itself SHALL be sent defining the subscription interval using "Date:" header as starting time and "If-Modified-Since:" as ending time. The HC proxy can compute Lifetime option by using that HTTP headers.

Due to the asynchronous nature of this exchange, the HC proxy willing to accept establishing a subscription SHOULD send an HTTP response with status "206 Partial Content", Content-Type "multipart/mixed" and Transfer-Encoding "chunked".

Each CoAP response will be delivered in a different HTTP chunk until the subscription lifetime expires, when the subscription has expired the HTTP session MUST be closed.

If the HC proxy does not support this exchange or is not willing to establish this session, it SHOULD fail with status "417 Expectation failed".



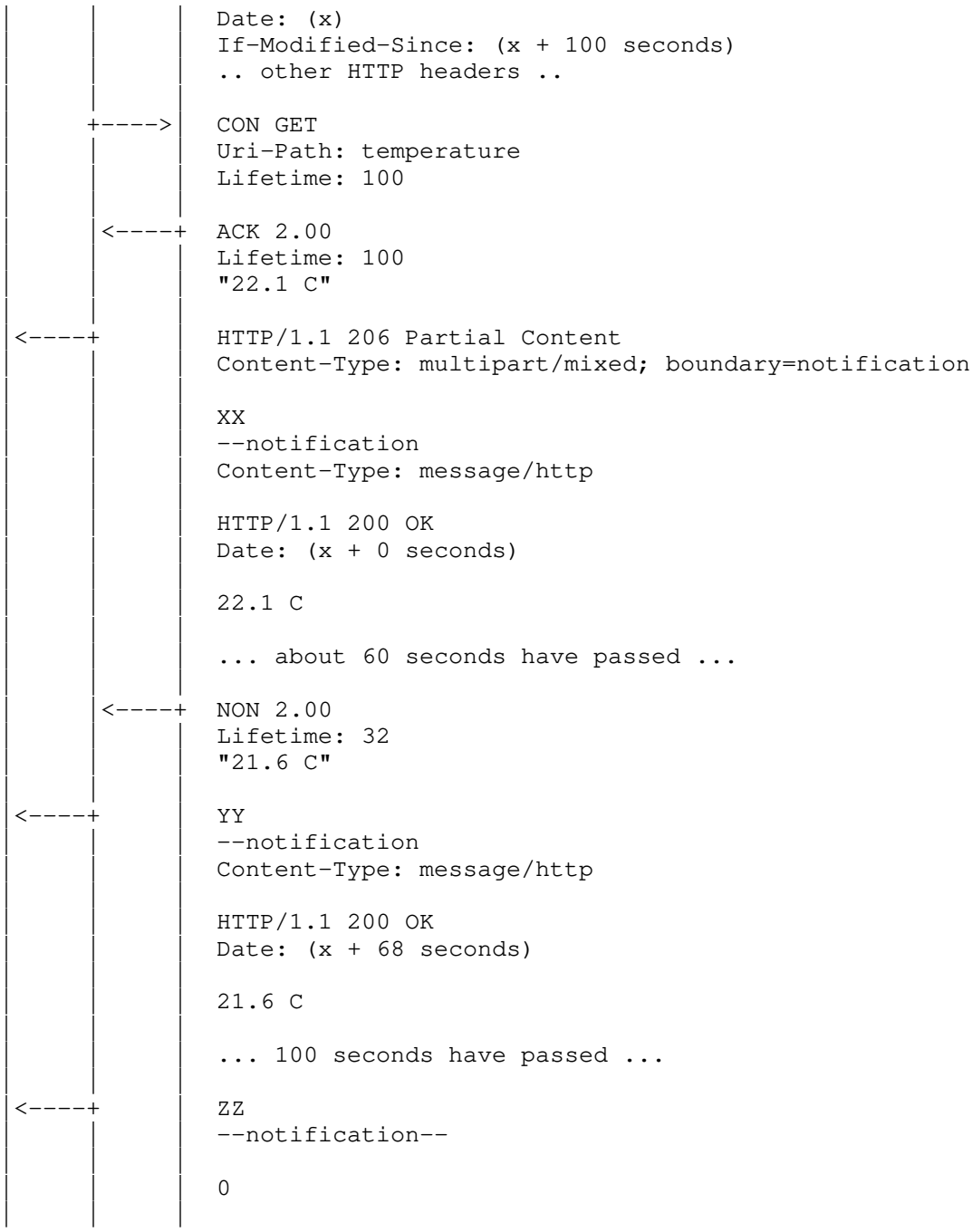

Date: (x)
If-Modified-Since: (x + 100 seconds)
.. other HTTP headers ..
CON GET
Uri-Path: temperature
Lifetime: 100
ACK 2.00
Lifetime: 100
"22.1 C"
HTTP/1.1 206 Partial Content
Content-Type: multipart/mixed; boundary=notification
XX
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 0 seconds)
22.1 C
... about 60 seconds have passed ...
NON 2.00
Lifetime: 32
"21.6 C"
YY
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 68 seconds)
21.6 C
... 100 seconds have passed ...
ZZ
--notification--
0

Figure 4: HTTP subscription to a CoAP resource

When an HTTP client performs direct subscriptions to CoAP servers using this method, the HC proxy has to keep for a possibly long time state information about the observe session and an open HTTP/TCP session to the client.

Soft state required by the various involved protocols (HTTP/TCP, CoAP/UDP) leads to scalability issues when an high number of direct subscriptions are established using the same HC proxy.

Moreover the HC proxy has an active role in the subscription process, thus if crashed or rebooted the subscription to the CoAP node will be lost.

HTTP clients in the real world usually implement notification mechanisms over HTTP using a technique called "Long Polling", an extensive description of this technique is available in Section 2 of [I-D.loreto-http-bidirectional]. A mapping using a "Long Polling" may be identified and can be preferred for longer sessions of observe.

3. CoAP-HTTP

TBD

4. Security Considerations

TBD

5. IANA Considerations

This document does not require any actions by the IANA.

6. Acknowledgements

Special thanks to Nicola Bui and Michele Zorzi for their support and for the various contributions to this document.

Thanks to Kerry Lynn, Akbar Rahman, Peter van der Stok and Anders Brandt for the extensive fruitful discussion about URI mapping, DNS and multicast group communication useful for writing various sections of this document.

Thanks to Brian Frank for its support and its feedback about the content.

7. References

7.1. Normative References

[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.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[I-D.ietf-httpbis-p1-messaging] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-12 (work in progress), October 2010.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-observe] Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-01 (work in progress), February 2011.

7.2. Informative References

[I-D.loreto-http-bidirectional] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", draft-loreto-http-bidirectional-07 (work in progress), January 2011.

[I-D.thomson-hybi-http-timeout] Thomson, M., Loreto, S., and G. Wilkins, "Hypertext Transfer Protocol (HTTP) Timeouts", draft-thomson-hybi-http-timeout-00 (work in progress), March 2011.

[I-D.jennings-http-srv] Jennings, C., "DNS SRV Records for HTTP",

draft-jennings-http-srv-05 (work in progress), March 2009.

[I-D.rahman-core-groupcomm]
Rahman, A., "Group Communication for CoAP",
draft-rahman-core-groupcomm-04 (work in progress),
March 2011.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com