Analysis of Constrained Application Protocol

Technical Report – FIT-TR-2017-15

Ondřej Ryšavý

Technical Report no. FIT-TR-2017-15
Faculty of Information Technology
Brno University of Technology Brno,
Czech Republic

December, 2017

Analysis of CoAP Protocol

1. Introduction

IoT nodes can be very lightweight in the sense of consumption, resources, and capabilities. To enable their communication the communication protocol should reflect the limitations of IoT nodes. While common Internet protocols can be used in IoT environment, new protocols addressing the need for IoT nodes emerges. In this report, we focus on Constrained Application Protocol (CoAP). It is one of the applications protocols of IoT protocol family being considered for standardization by IETF that also comprises MQTT, DDS, AMQP, XMPP and HTTP REST.

While there are some attempts to propose a unifying architecture for IoT, many architectural models have been designed and applied. Common architectures for IoT are represented as three-layered, middleware based, or five-layered. CoAP is an application protocol for IoT environment and can be adapted to all mentioned IoT layered architectures (see Figure 1).

CoAP protocol was designed for communication in the environment of constrained nodes. The protocol is not intended to replace HTTP. Interoperability with typical Internet environment is often necessary for CoAP deployment. Typical interoperability scenario involves CoAP-HTTP gateway. The constrained realm of IoT devices uses CoAP application protocol to exchange data. When necessary to access IoT devices from the Internet the gateway performs necessary translations between internet application protocol (HTTP) and IoT application protocol (CoAP). While CoAP can be used in the Internet environment too, HTTP is usually preferred as it offers more robust and reliable data communication.

2. CoAP Protocol

Constrained Application Protocol (CoAP) is an application layer protocol for constrained communication in IoT applications. In principle, it follows REST paradigm and message semantics similar to HTTP but with lower overhead and some extensions suitable for IoT environment. CoAp can be carried in UDP while HTTP requires reliable transport offered by TCP. The advantage of adopting REST approach is in making CoAP interoperability with REST HTTP straightforward.

Because of no reliable data transport is required, CoAP needs to solve the problem of reliable data communication. It specifies messaging sub-layer that detects duplications and assures reliable data delivery. The request-response layer then implements REST communications. CoAP also

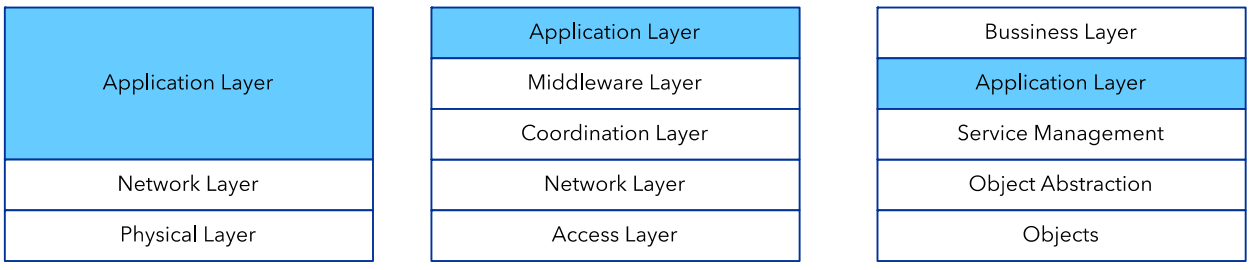


Figure 1: IoT Architecture Models

provides some advanced features such as native push notifications and group communication. The security of data communication is ensured by using DTLS, which is an adaptation of TLS for UDP protocol.

2.1. Resource Naming

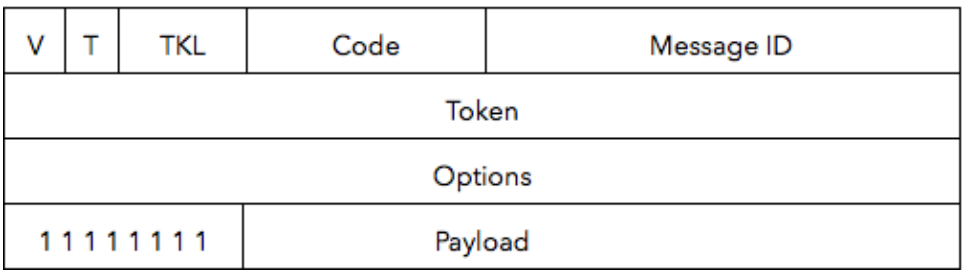
CoAP uses URI scheme to identify resources. URI is split in CoAP header into options each carrying part of the complete URI. Because of this, only the URI component can be carried in CoAP message saving the valuable space in CoAP message. URI for CoAP consists of four parts:

- (“coap:”) - the scheme used for resource access that always is CoAP protocol.
- (“//” Uri-Authority) – The host needed to localize the resource; this is especially important when virtual servers are deployed on a single host (same as in HTTP). In cases, when the resource can be localized by IP address this can be omitted.
- (“/” Uri-Path) – absolute path of the resource, because of constrained environment, short path descriptors should be used.
- (“?” Uri-Query) -When necessary this part represents a query part of the URI; that is a series of variable value pairs in the form of “key=value.”

2.2. Messages

The format of CoAP message is depicted in Figure 2. CoAP message contains a small header of fixed fields followed by options and payload. Options use a special kind of TLV encoding explained later and demonstrated in Figure 4. The fixed header consists of the following parts:

- V – Version (2-bits). The current version is version 1.
- T – Type of CoAP message (2-bits). CoAP uses four types of messages: Confirmable (0), Non-confirmable (1), Reset (2), and Acknowledgment (3).
- TKL - 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes).
- Code – Method or Response Code (8-bits) of a message determines the meaning of the message. Common method codes are GET (1), POST (2), PUT (3), DELETE (4). Response codes are in the interval 40-255 and denote a success of the operation or characterize the error.
- Message ID – unsigned 16-bit integer stands for an identification of request-response pair of messages. MID is essential for reliable communication. MID must be fresh for new requests but must not change for retransmission of an existing request.
- Token, if any, may be 0 to 8 bytes as given by the Token Length field. The Token value is used to correlate requests and responses.



Options sections can be empty. However, for most of the operations, some options are required. Options start with the type specification followed by length and value. Type specification is only four bits. To provide more than 16 different option types, this field does not directly represent an option type, but rather it is an increment to the previous option type. Thus the option type can be obtained by summing all option type fields up to the current option. For instance, the sequence of option items with type values 1,4,4,2,4 represents the following items:

Type Value	Option Type	Option Name
1	1	Content-type
4	5	Uri-Authority
4	9	Uri-Path
2	11	Token
4	15	Uri-Query

Some options can have a variable length. The maximum size of the option value is limited to 270 bytes because of the size of the length field. Specific TLV encoding is employed to save space in the options section as explained in Error! Reference source not found.. Type part is a 4-bit value representing an increment to previous option item. To compute the type, this value is added to the previous option. For the first option, it directly represents the option id. Options thus have to be ordered according to their IDs in the CoAP message. Length part is another 4-bits when the length of option value is less than 15 bytes. If the length of option's value is between 15 and 270 octets length is encoded in the second byte.

2.3. Request/Response Model of Communication

The CoAP basic style of communication is based on Request/Response Model. The client sends a request message to a server and the server answers with a response message. The character of response depends on the requested Type and Code.

Some fields carry valuable information for matching request response messages and for identifying transactions.

- Message ID is used to identify which response match the request message. Response to the specific request MUST have the same TID. Usually, the request has CON type, and the response of ACK type is expected.
- Token is an option that is involved in non-trivial communication. Message ID determines only a pair of messages. If a transaction involves multiple message pairs, the token identifies all these messages as a part of the transaction.

Figure 2: CoAP Message

Some fields are critical to CoAP communication. A CoAP request message consists of the following mandatory fields:

Field	Meaning
ID	Represents a message ID. The requestor sets it for Confirmable and Non-Confirmable messages. This ID is used to match an Ack messages.
Type	Represents a message type, which is one of the following: CON – confirmable message requires acknowledgment. NON – non-confirmable message does not require acknowledgment.
Token	The token option is used to distinguish further among concurrent request/response transactions.
MethodCode	Basic methods of GET, POST, PUT and DELETE with a similar meaning is in HTTP.
URI	A resource identifier is encoded using three separate options: • The Uri-Authority Option indicates the authority (host + port) part of a URI, and conforms to "host [: port]." • The Uri-Path Option indicates the absolute path part of a URI. • The Uri-Query Option indicates the query part of a URI (if any).
ContentType	Content-type determines application payload.
Payload	The payload can be any data in binary or text formatting. Content-type option determines the type of data.

A CoAP response message consists of the following major fields:

Field	Meaning
ID	For ACK response this field should match to the corresponding CON message.
Type	Represents a message type, which is one of the following: ACK – message that acknowledges received confirmable message RST – answer to CON message that cannot be processed because of missing context
Token	The token option is used to correlate messages in concurrent or asynchronous transactions.
ResponseCode	The status code of the transaction. The most often used codes are OK (80), Created (81), Not Modified (124), Bad Request (160), Not Found (164) and Not Allowed (165).
ContentType	Content-type determines application payload. Common media types are text/plain (0), text/xml (1), image/gif (21), image/jpeg (22), application/link-format (40), application/octet-stream (42), application/json (51).
Payload	The payload can be any data in binary or text formatting. Content-type option determines the type of data.

2.4. Reliable Delivery

Transactions comprise of message exchange between clients and servers that have the same token. When reliable communication is required, CoAP protocol uses CONFIRMABLE message type to inform the remote party that ACKNOWLEDGEMENT is required. If the ACKNOWLEDGEMENT message is not received within predefined period, the CONFIRMABLE message is resent.

Depending on whether the result can be immediately provided or not there are two possible scenarios as explained in Figure 4.

- If the result is available, the response is sent immediately in ACK message answer. This is called piggybacked response.

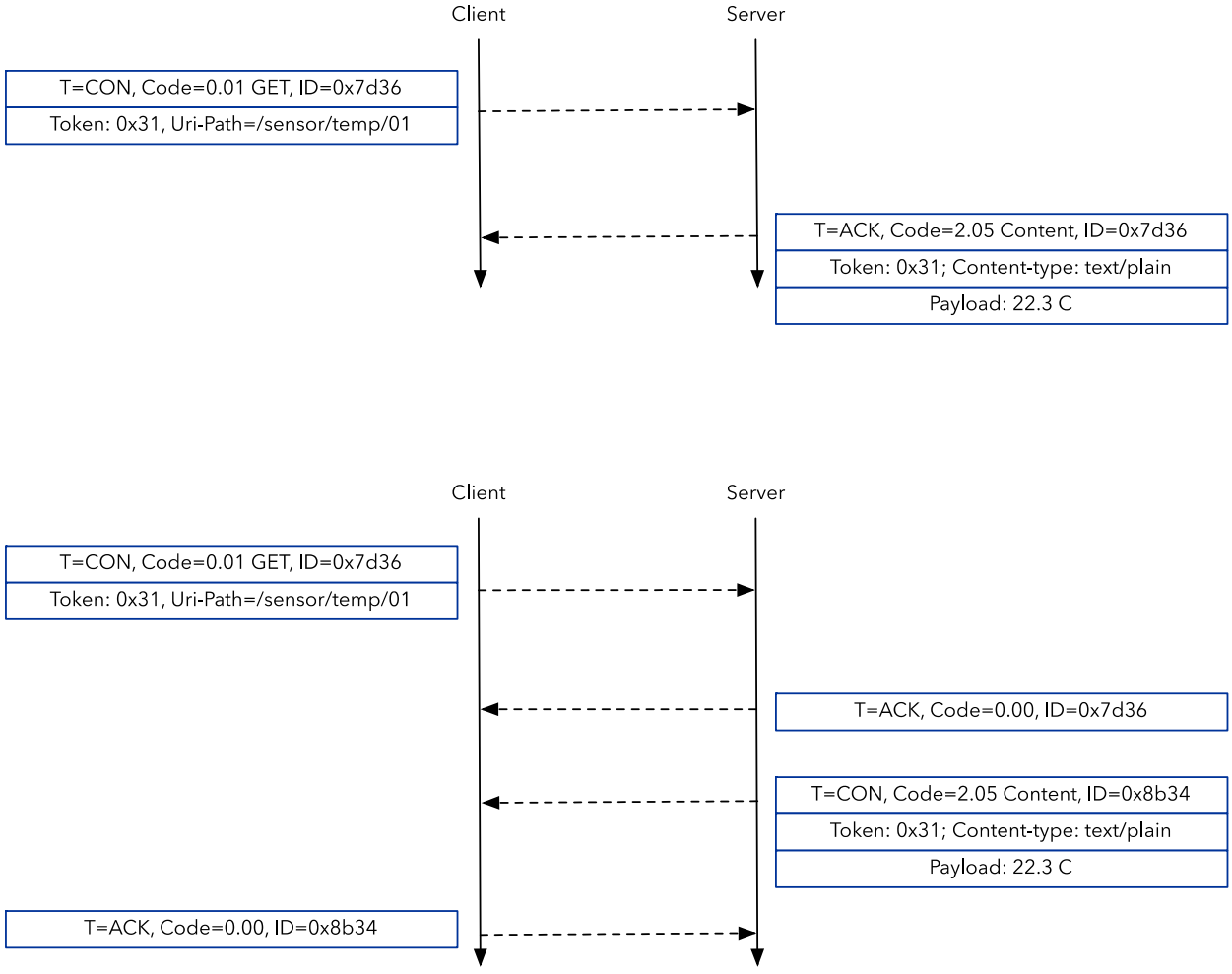


Figure 3: Reliable Transactions

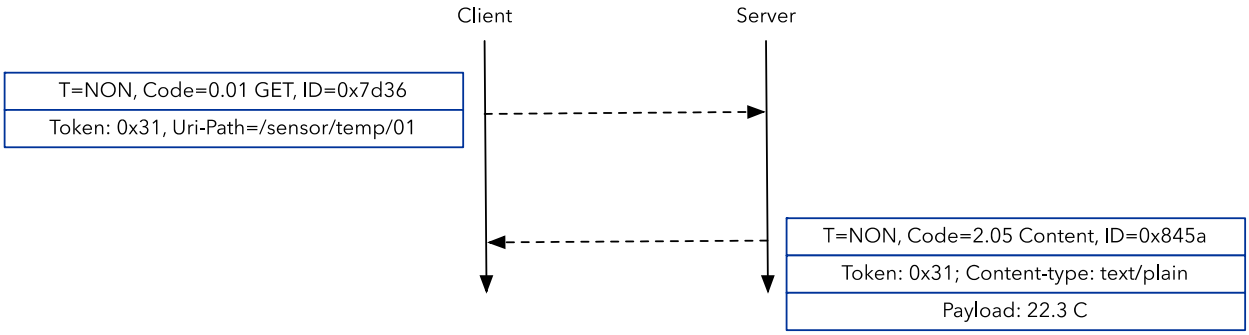


Figure 4: Unreliable Transaction

- If it is not possible to provide result immediately, then the response is sent in a separate message. It means that the server sends ACK response without the required result and when the result is available it is sent in a separate message. Figure 5 shows piggybacked and separated transactions.

Observe ID of messages. In piggybacked case, both messages have the same ID. In separated case, there are two pairs of messages. In this case, the token value is used to link these two pairs together as a separated transaction.

2.5. Unreliable Transactions

Non-confirmable communication is simpler as the communicating parties do not require reliable message delivery. The client sends a request in NONCONFIRMABLE message type. The server also replies with the NONCONFIRMABLE message type. Note that ID of the messages may be different as the response does not acknowledge the request message. The correlation of these two messages is based on the same token value. Figure 5 shows an example of non-confirmable communication.

2.6. Advanced Features

CoAP protocol was extended with some advanced features to support various needs of IoT environments. Observers are used in scenarios when a client wants to be informed about the changes of the target resource. This scenario can be realized by polling mechanism implemented as usual request/response message exchange. Observes extension simplifies this scenario enabling clients to register at the target resource for receiving notifications.

Another extension is the experimental support for group message delivery. The CoAP group communication is realized by a single multicast request and multiple unicast responses.

Resource Discovery enables clients to enumerate resources available at the server. Resource discovery is built on the top of URI capabilities. Discovering resources on a specific host is performed by sending a GET request to “/.well-known/core” which is defined as a default entry-point. Server’s respond then should list available resources.

Blockwise Transfer extension was introduced to enable high volume data transfer. The original design of CoAP considered that data exchanged in messages is small and fits in a single message in most cases. For software updates or downloading large information the usually request-response message exchange is inefficient.

2.6.1. Observers

An extension of CoAP with the ability to observe resources by implementing publish/subscribe mechanism of communication is proposed in RFC7641 [2]. Instead of repeated queries, the client registers itself to the server and receives updates for interested resources. As this would incur overhead, the protocol provides a best-effort delivery of updates thus the eventual consistency model is only guaranteed.

The protocol implements the observer design pattern. Observers register themselves at providers called subjects. A subject informs registered observers when the state changes. In the context of CoAP the key elements have the following meaning:

- The subject is a resource located on a CoAP server.
- The observer is a CoAP client that is informed about subject's state change.
- Registration is a GET request sent by the observer to the CoAP server.
- Notification is CoAP response sent by the server that includes the new resource state.

Registration and Notification messages exchanged between server and observer use option Observe to identify the order of updated records. The token value is employed to correlate notifications to corresponding registrations. Observe option is used to determine the order of a notification message. See Figure 6 for example of observer registration and a sequence of notifications.

A lifetime of each registration is determined by the interest of the observers by the combination of the following methods:

- Observer sends a Deregister message to the server.
- Observer rejects (RST message) notification.

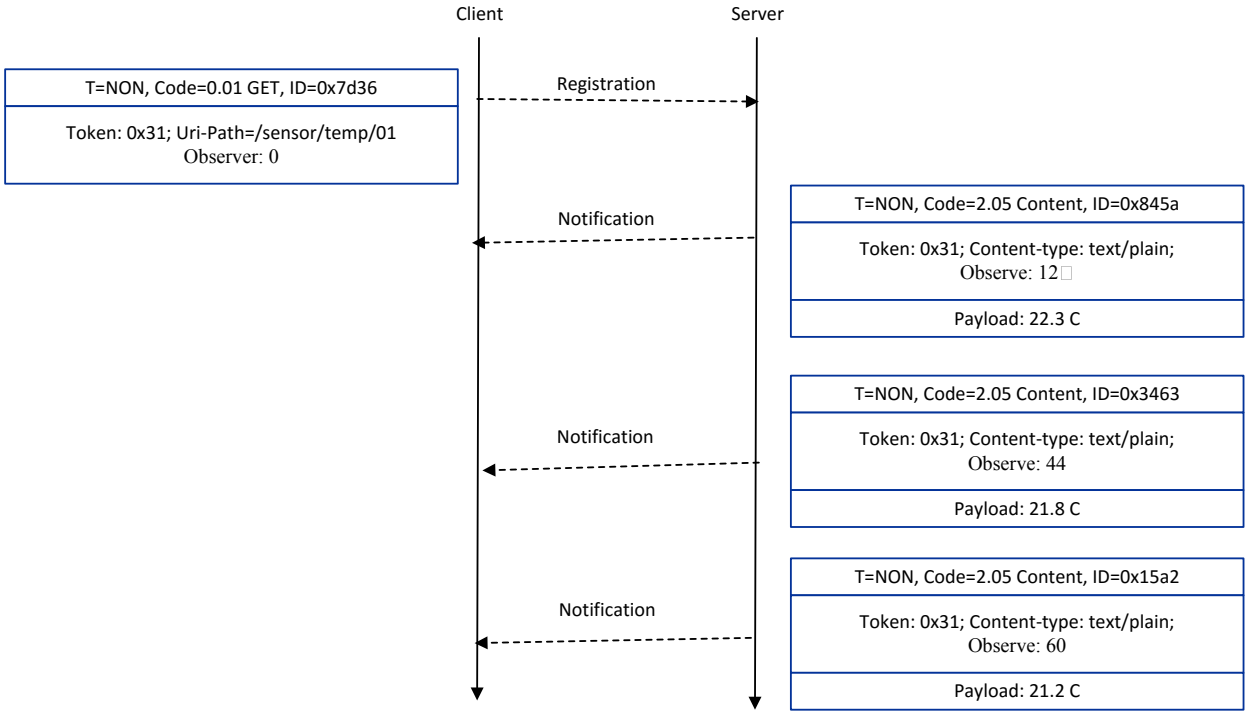


Figure 5: Registration and Notifications

- The observer does not acknowledge the confirmable notification.

The protocol provides only eventual consistency model. It is because of the following:

- Latency between the change of the state and reception of the notification.
- Notification can get lost as best-effort is used for informing observers.
- The server may incorrectly consider that the observer is not further interested in the notification.

The protocol itself does not specify when and how often the server sends notifications. It depends on the server implementation. However, it is possible to parametrize resources' URLs to enable specifying trigger value, for instance: <coap://server/temperature/critical?above=42>.

Observe option has ID=6 and its length is up to 4 bytes. The meaning of the Observe value depends on whether it is used in request or response. If Observe is in GET request then the value means:

- 0 (register) adds the entry to the list, if not present;
- 1 (deregister) removes the entry from the list if present.

If Observe is a part of the notification, the option value is a sequence number for reordering detection. The value of the Observe Option is encoded as an unsigned integer in network byte order using a variable number of bytes.

2.6.2. Multicast Group Communications

The CoAP provides for group communications and multicast as specified in experimental RFC7390 [3]. Group communication in IoT is common. For example, CoAP control node may switch on/off multiple devices with a single group request. The multicast capability of IP protocol is considered for CoAP group communication. The CoAP group communication is realized by a single multicast request and multiple unicast responses. The assumption is that the source node may or may not be a part of the group and also that there may be more source nodes. A CoAP group consists of a collection of CoAP endpoints that are members of the same IP multicast group. To enable CoAP discovery, all nodes should be members of "All CoAP Nodes" group: 224.0.1.187. Fro IPv6, CoAP nodes should join link-local scoped address ff02::fd and the site-local scoped address ff05::fd. Uri authority should contain a group IP or Group Fully Qualified Domain Name that can be resolved to the group IP multicast address, for instance, *all.west.bldg6.example.com*. Sending group messages requires specific considerations:

- CoAP group communication does not work if different ports are used on group's nodes.
- Idempotent methods can be safely used with multicast. For non-idempotent methods, such as DELETE and POST, the operations should be designed to take message loss into account.
- CoAP group communication uses only non-confirmable messages. Optional unicast responses may not be complete depending on whether some messages were lost. Rules for reusing token values are more complicated comparing to unicast communication.

A set of operations related to membership configuration are necessary to properly maintain CoAP groups:

- Member discovery can be performed with the help of Resource Directory [6] or use some proprietary management system.

- The configuration of a group membership of a CoAP node can be (i) preconfigured before deployment, (ii) set up by executing discovery procedure, or (iii) configured from another node.
- CoAP endpoints implementing the membership configuration RESTful interface must support the CoAP group configuration Internet Media Type "application/coap-group+json".

2.6.3. Resource Discovery

Resource Discover in CoAP environment is essential as this domain often lacks human administrator that could provide information on resource localization. Resource discovery is a mechanism that assigns URI for resources host by CoAP servers. Resource discovery is thus built on the top of URI capabilities. Discovering resources on a specific host is performed by sending a GET request to "/.well-known/core" which is defined as a default entry-point. The response to this request is a list of all available resources. To obtain only a filtered subset of all available resources the query part of the request is used, for instance, /.well-known/core?rt=temp to get resources with temp in their name. In addition to basic discovery scenario, in which resources of the single CoAP node are discovered, it is possible to discover resource collections. It can be achieved, for instance, using Resource Directory [6] interface.
CoAP Resources Discovery mechanism uses Web Links that were designed to indicate the relationships between resources on the Web. A Link is a typed connection between two resources defined as:

```
"<" TARGET-URI ">" ( ";" LINK-PARAM )*
```

Where:

- TARGET-URI – each link contains one target URI inside angle brackets. The target URI must be a relative URI of the context URI that is determined by the CoAP endpoint's authority URI.
- LINK-PARAM – specific target attributes are defined for CoAP resources. For instance, (i) the resource type "rt" is an opaque string used to assign an application-specific semantic type to a resource, or (ii) the interface description "if" attribute is an opaque string used to provide a name or URI indicating specific interface definition that can be used to interact with the resource.

The following example presents two links with target URI "sensors/temp" and "sensors/light", respectively. Both links have only a single target attribute that states they have the same interface identifies as "sensor".

```
CON [0xaf6] GET /.well-known/core


ACK [0xaf6] 2.05 Content
```

```
</sensors/temp>;if="sensor",
</sensors/light>;if="sensor"
```

Consider that context URI (URL of the CoAP server) is coap://server.domain.org then URIs of the resources are:

```
coap://server.domain.org/sensors/temp
coap://server.domain.org/sensors/light
```

2.6.4. Blockwise Transfer

CoAP was designed for constrained environment where most of the traffic consists of small chunks of data. However, some scenarios require transferring larger payloads. RFC7252 introduces a block-wise mode of communication as a method of transferring multiple blocks of information from a resource representation in multiple request-response pairs. Block-wise transfer aims at avoiding the fragmentation for segments larger than maximum PDU size of the underlying protocol. Also, this mechanism was designed such that the conversation state is not necessary.

In general, the block-wise transfer is used to send data larger than PDU size of supporting protocols. Two new options are introduced to support block-wise transfer: Block1 (option number 27) and Block2 (option number 23) are options that can be used in request and response. Depending on their occurrence they have slightly different interpretation:

- Block1 in the *request* or Block2 in *response* describes how the block-wise payload forms part of the entire body being transferred ("descriptive usage").
- Block1 in *response* or Block2 in a *request* provides additional control on how that payload will be formed or was processed ("control usage").

Block option contains three items:

- The size of the block (SZX) – three-bit uint encoding size exponent, the size of the block is computed as 2**(SZX+4) bytes.
- Whether more blocks are following (M) - single bit.
- The relative number of the block (NUM) within a sequence of blocks with the given size. This item has a variable size between 4 and 19 bits.

All these items are encoded using a variable-size uint type. An example of blockwise data transfer is in Figure 7: Blockwise data transfer.

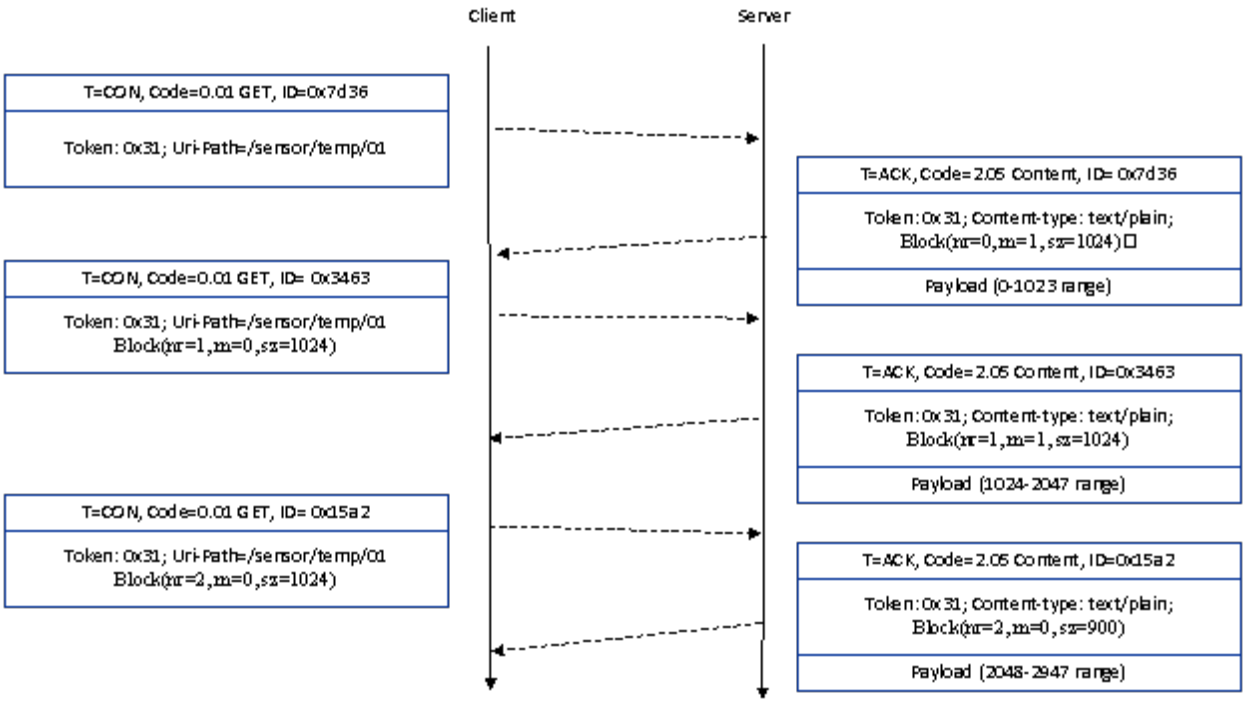


Figure 6: Blockwise data transfer

3. Exported CoAP Fields

This section summarizes CoAP fields and options. This specification was adapted from Wireshark display filter reference[1]. Exported information from CoAP message can be a subset of the presented items.

Field Name	Type	Description
coap.code	UINT8	Method or Response Code (8-bits) of a message determines the meaning of the message.
coap.mid	UINT16	Message ID.
coap.ocount	UINT8	Option Count.
coap.payload	BINARY	Payload of CoAP message
coap.tid	UINT16	Transaction ID.
coap.type	UINT8	Type of CoAP message, e.g., Confirmable, Nonconfirmable, Reset, Ack.
coap.version	UINT8	A version of CoAP message.

CoAP options are split into groups depending on their meaning.
General options are as follows:

Field Name	Type	Description

[1] https://www.wireshark.org/docs/dfref/c/coap.html

coap.opt.accept	STRING	Specifies the acceptable content type of the response.
coap.opt.ctype	STRING	Specifies the content type of the payload.
coap.opt.token	BINARY	Token content. This has a variable size.
coap.opt.token_len	UINT8	Token field length.

Uri-related options are used for describing the requested resource:

Field Name	Type	Description
coap.opt.uri_auth	STRING	Authority part of the URI.
coap.opt.uri_host	STRING	The host part of the URI.
coap.opt.uri_path	STRING	Path part of the URI.
coap.opt.uri_port	UINT16	Port part of the URI.
coap.opt.uri_query	STRING	Query part of the URI.

Observers extension and block-wise transfer use the following options:

Field Name	Type	Description
coap.opt.block_mflag	BOOL	More blocks flags.
coap.opt.block_number	UINT32	A sequence number of the block.
coap.opt.block_size	STRING	Path part of the URI.
coap.opt.observer	UINT32	Sequence number of the notification message.

4. CoAP Events

This section provides a list of CoAP events identifiable by analyzing CoAP communication. An event is emitted when the specific communication pattern is identified. For most of the events, it is enough to track message ID and Token option.

Event name	Explanation
coap.event.request	Emitted on the occurrence of a CoAP request message.
coap.event.response	Emitted on the occurrence of a CoAP response message.
coap.event.simple_transaction	Emitted when a single CoAP transaction occurs. This happens if the request message and the corresponding response message were processed.
coap.event.duplicate_request	A duplicate request message was found. This occurs when two request messages of the same message ID were processed.
coap.event.duplicate_response	A duplicate response message was found. This occurs when two response messages of the same message ID were processed.
coap.event.block_transaction	Emitted when a blockwise transfer completes.
coap.event.split_transaction	Emitted when a sperated transaction completes.

coap.event.observer_register	Emitted when an observer sends a register message to the resource owner.
coap.event.observer_deregister	Emitted when an observer sends a deregister message to the resource owner.
coap.event.observer_notification	Emitted when notification message.
coap.event.resource_enum	Emitted when resource discovery request message was identified.

References

1. Shelby, Z., Hartke, K., & Bormann, C. (2014). The Constrained Application Protocol (CoAP). IETF RFC 7252.
2. Hartke, K. (2015). Observing Resources in the Constrained Application Protocol (CoAP). IETF RFC 7641. Retrieved from https://tools.ietf.org/html/rfc7641
3. Rahman, A., & Dijk, E. (2014). Group Communication for the Constrained Application Protocol (CoAP). IETF RFC 7390. Retrieved from https://tools.ietf.org/pdf/rfc7390.pdf
4. Shelby, Z. (2012). Constrained RESTful Environments (CoRE) Link Format. IETF RFC 6690. Retrieved from https://tools.ietf.org/html/rfc6690
5. Bormann, C. & Sheldby, Z. (2016). Blockwise transfers in the Constrained Application Protocol (CoAP). IETF RFC 7959. Retrieved from https://tools.ietf.org/html/rfc7959
6. Shelby, Z., Koster, M., Bormann, C., van der Stok, P. (2016) CoRE ResourceDirectory. IETF Draft. Retrieved from https://tools.ietf.org/html/draft-ietf-core-resource-directory-09