DOI 10.12694/scpe.v14i4.931
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 249–263. http://www.scpe.org

A SESSION INITIATION PROTOCOL FOR THE INTERNET OF THINGS *

SIMONE CIRANI, MARCO PICONE AND LUCA VELTRI†

Abstract. The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects", in an Internet-like structure. Smart objects typically feature limited capabilities in terms of computation and memory and operate in constrained environments, such as low-power lossy networks. As the Internet Protocol (IP) has been foreseen as the standard for communications in IoT, an effort to bring IP connectivity to smart objects and define suitable communication protocols (i.e. Constrained Application Protocol (CoAP)) is being carried out within standardization organizations, such as the Internet Engineering Task Force (IETF). In this paper, we propose a constrained version of the Session Initiation Protocol (SIP), named "CoSIP", whose intent is to allow constrained devices to instantiate communication sessions in a lightweight and standard fashion. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication. CoSIP can be adopted in several application scenarios, such as service discovery and publish/subscribe applications, which are detailed. An evaluation of the proposed protocol is also presented, based on a Java implementation of CoSIP, to show the benefits that its adoption can bring about, in terms of compression rate with the existing SIP protocol and message overhead compared with the use of CoAP.

Key words: Internet of Things, communication protocols, CoAP, SIP, signaling, service discovery

1. Introduction.

The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects", in an Internet-like structure. Smart objects have limited capabilities, in terms of computational power and memory (e.g., 8-bit microcontrollers with small amounts of ROM and RAM), and might be battery-powered devices, thus raising the need to adopt particularly energy efficient technologies. Smart objects typically operate in constrained networks which often have high packet error rates and a throughput of tens of kbit/s. In order to interconnect smart objects in an Internet-like architecture, standard and interoperable communication mechanisms and protocols are required, and a lot of work is currently ongoing for defining proper standards. It is a common opinion that in the near future IP will be the base common network protocol for the IoT. This does not imply that all objects will be able to run IP. In contrast, there will always be tiny devices, such as tiny sensors or Radio-Frequency IDentification (RFID) tags, that will be organized in closed networks implementing very simple and application-specific communication protocols and that eventually will be connected to an external network through a proper gateway. However, it is foreseen that all remaining small networked objects will exploit the benefits of IP and corresponding protocol suite.

In [1], the author tries to define the following pair of classes for constrained devices, in terms of memory capacity, in order to be used as a rough indication of device capabilities:

- class 1: RAM size = ∼10 KB, Flash size = ∼100 KB
- class 2: RAM size = ∼50 KB, Flash size = ∼250 KB

Some of these networked objects, with enough memory, computational power, and power capacity, will simply run existing IP-based protocol suite implementations. Some others will still run standard Internet protocols, but may benefit from specific implementations that try to achieve better performance in terms of memory size, computational power, and power consumption. Instead, in other constrained networked scenarios, smart objects may require additional protocols and some protocol adaptations in order to optimize Internet communications and lower memory, computational, and power requirements.

As billions of smart objects are expected to come to life and IPv4 addresses have eventually reached depletion, IPv6 [2] has been identified as a candidate for smart-object communication at the network layer. However, due to the possible limitations that constrained devices may encounter, some adaptation at network layer (IP) and at upper layers may be required.

*The work of Simone Cirani and Luca Veltri has been funded by the European Community's Seventh Framework Programme, area "Internetconnected Objects", under Grant no. 288879, CALIPSO project - Connect All IP-based Smart Objects. The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein.

†Department of Information Engineering, University of Parma, Viale G.P. Usberti, 181/A, 43124 Parma, Italy (simone.cirani, marco.picone, luca.veltri@unipr.it).

Significant reasons for proper protocol optimizations and adaptations for resource-constrained objects can be summarized as follows.

- Smart objects typically use, at physical and link layers, communication protocols (such as IEEE 802.15.4) which are characterized by small Maximum Transmission Units (MTUs), thus leading to packet fragmentation. In this case, the use of compressed protocols can significantly reduce the need for packet fragmentation and postponed transmissions.
- Processing larger packets likely leads to higher energy consumption, which can be a critical issue in battery-powered devices.
- Minimized versions of protocols (at all layers) can reduce the number of exchanged messages.

For this reason, within the IETF some specific working groups have been set. In particular, the IETF 6LoWPAN (IPv6 over Low power WPAN) Working Group [3] is defining encapsulation and other adaptation mechanisms to allow IPv6 packets to be sent to and received from over Low power Wireless Personal Area Networks, such as those based on IEEE 802.15.4. for the network layer, the ROLL (Routing Over Low power and Lossy networks) Working Group [4] is currently studying and defining proper routing mechanisms. Instead, for the application layer, the IETF CoRE (Constrained RESTful Environments) Working Group [5] is currently defining a Constrained Application Protocol (CoAP) [6], to be used as a generic web protocol for constrained environments, targeting Machine-to-Machine (M2M) applications, and that can be seen in some ways as a compressed version of the HyperText Transfer Protocol (HTTP) [7]. CoAP will include the following features:

- request/response interaction model between application endpoints;
- built-in discovery of services and resources;
- key concepts of the Web such as URIs (Uniform Resource Identifiers) and Internet media types.

The typical Internet of Things protocol stack, compared with the standard web protocol stack, is depicted in Fig. 1.1. The symmetry between the the two protocol stacks is clear: for instance, at the application layer, while HTTP is the most widespread application protocol for Internet applications, such as the World Wide Web, its constrained version, the CoAP, is expected to be used, reducing the complexity of implementation as well as the size of packets exchanged. CoAP, in contrast to HTTP, uses UDP [8] as a lightweight transport protocol.

CoAP is intended to provide application a RESTful (Representational state transfer) communication mechanism. According to the REST model, representations of resources are exchanged between a client and a server. A resource representation is the current or intended state of a resource referred to the server through a proper namespace. A client that is interested in the state of a resource sends a request to the server; the server then responds with the current representation of the resource. An example of CoAP request/response interaction is depicted in Fig. 1.2.

If the client is interested in receiving the representation during a period of time, according to this model the client should periodically repeat such requests in order to always have the updated representation of the resource. Of course this mode of operation would lead to unnecessary messages sent over the network and processing load at both client and server sides, each time a request is sent for a resource state that is not changed. In order to save both network and processing resources, a more suitable communication model, following the event subscribe/notify paradigm, can be used. According to this paradigm, a client, called "observer", interested in a resource subscribes to a resource, called "subject", informing the server that it is interested in being notified whenever the subject changes the state; we say that the clients want to "observe" the resource.

There is a CoAP extension [9] currently in the IETF standardization process, that aims at defining a CoAP-based subscribe/notify service can be implemented over the basic CoAP REST model. Figure 1.3 compares the two approaches of "observing" a resource.

However, beside REST and subscribe/notify service models, there are also many other applications in both constrained and non-constrained environments that feature non-request/response communication models. Some of these applications require the creation and management of a "session". With the term of "session" we refer to any exchange of data between an association of participants. In case of two participants, the session may involve the sending of one or (probably) more data packets from one participant to the other, in one or both directions. In case of unidirectional sessions, they may be initiated by both the sender or the receiver. Examples of sessions in IoT scenario may be the data flow generated by a sensor (measurement samples) and sent to a

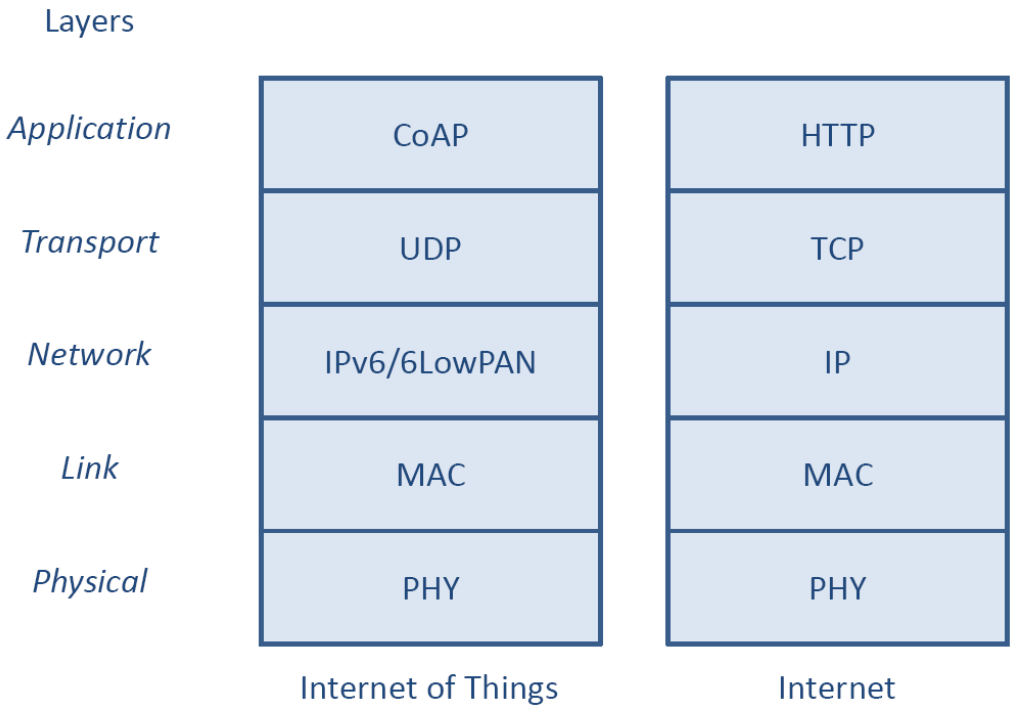

FIG. 1.1. *Comparison between the IoT and the Internet protocol stack for OSI layers.*

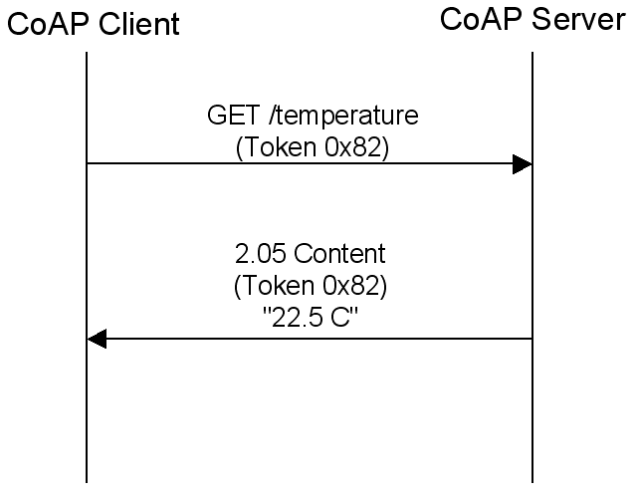

FIG. 1.2. *CoAP request/response model.*

given recipient for further processing, or some data streams exchanged by two interacting toys.

Although in principle CoAP encapsulation could be used also for carrying data in non-request/response fashion, for example by using CoAP POST request in non-confirmable mode, or by using the CoAP "observation" model, it is evident that could be much more efficient to setup a session between constrained nodes first, and then perform a more lightweight communication without carrying unnecessary CoAP header fields for each data packet. The data communication will be in accord to the network, transport, and application parameters negotiated during the session setup.

The Session Initiation Protocol (SIP) [10] is the standard application protocol for establishing application-

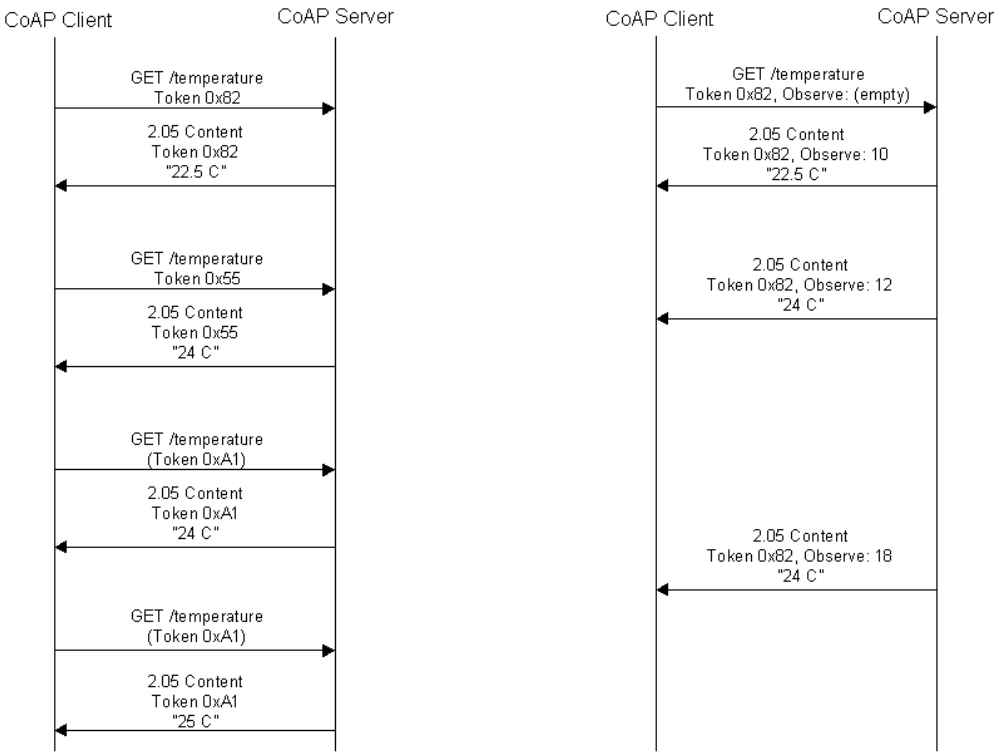

FIG. 1.3. *CoAP basic model (left) vs. CoAP "observing" (right). The CoAP "observe" extension introduces a subscribe/notify communication model, in contrast to the request/response model of standard CoAP.*

level sessions. It allows the endpoints to create, modify, and terminate any kind of (multi)media sessions, such as VoIP calls, multimedia conferences, or data communications. Once a session has been established, the media are transmitted typically by using other application-layer protocols, such as RTP and RTCP [11], or as raw UDP data, directly between the endpoints, in a peer-to-peer fashion. SIP is a text protocol, similar to HTTP, which can run on top of several transport protocols, such as UDP (default), TCP, or SCTP, or on top of secure transport protocol such as TLS and DTLS. Session parameters are exchanged as SIP message payloads; a standard protocol used for this purpose is the Session Description Protocol [12]. The SIP protocol also supports intermediate network elements which are used to allow endpoint registration and session establishment, such as SIP Proxy servers and Registrar servers. SIP also defines the concepts of transaction, dialog, and call as groups of related messages, at different abstraction layers.

Although SIP has been defined for Internet application, we may think to re-use it also in constrained IoT scenario. Note that SIP already includes mechanisms for subscribe/notify communication paradigms [13] and for resource directory, particularly useful in IoT scenarios, for which proper CoAP extensions are currently being specified [9, 14].

The main drawback of using standard SIP protocol in constrained environments is the large size of text-based SIP messages (compared to other binary protocols such CoAP), and the processing load required for parsing such messages.

For this reason, in this paper, we propose a constrained version of the Session Initiation Protocol, named "CoSIP", whose intent is to allow constrained devices to instantiate communication sessions in a lightweight and standard fashion and can be adopted in M2M application scenarios. Session instantiation can include a negotiation phase of some parameters which will be used for all subsequent communication. The proposed CoSIP is a binary protocol which maps to SIP, similarly to CoAP does to HTTP. CoSIP can be adopted in several application scenarios, such as service discovery and publish/subscribe applications.

The rest of this paper is organized as follows. In Section 2, an overview of related works is presented. In Section 3, fundamentals of the SIP protocol are summarized. In Section 4, the proposed CoSIP protocol is detailed together with its architecture and preliminary implementation. Some use cases for CoSIP-based applications in Internet of Things scenarios are presented in Section 5 and a performance evaluation of the proposed protocol is shown in Section 6. Finally, in Section 7 we draw our conclusions.

2. Related Work. Smart objects typically are required to operate using low-power and low-rate communication means, featuring unstable (lossy) links, such as IEEE 802.15.4, usually termed Low-power Wireless Personal Area Networks (LoWPANs) or Low-power and Lossy Networks (LLNs). The Internet Engineering Task Force (IETF) has setup several working groups in order to address many issues related to bringing IP connectivity to LoWPAN smart objects. In particular, the 6LoWPAN (IPv6 over Low power WPAN) WG [3] was chartered to work on defining mechanisms that optimize the adoption of IPv6 in LoWPANs and the ROLL (Routing Over Low power and Lossy networks) WG [4] was chartered to develop optimal IPv6 routing in LLNs. Finally, the CoRE (Constrained RESTful Environments) WG [5] has been chartered to provide a framework for RESTful applications in constrained IP networks. The CoRE WG is working on the definition of a standard application-level protocol, named CoAP, which can be used to let constrained devices communicate with any node, either on the same network or on the Internet, and provides a mapping to HTTP REST APIs. CoAP is intended to provide, among others, Create-Read-Update-Delete (CRUD) primitives for resources of constrained devices and publish/subscribe communication capabilities. While the work on CoAP is already at an advanced stage, the CoRE WG is also investigating mechanisms for discovery and configuration, but the work on these issues is still at an early stage and therefore open to proposals.

The "observer" CoAP extension [9] allows CoAP clients to observe resources (subscribe/notify mechanism) and be notified when the state of the observed resource changes. This approach requires the introduction of a new CoAP *Observe* option to be used in GET requests in order to let the client register its interest in the resource. The server will then send "unsolicited" responses back to the client echoing the token specified by the client in the GET request and reporting an Observe option with a sequence number used for reordering purposes. As we will describe later, we envision that the instantiation of a session could significantly reduce the amount of transmitted bytes, since, after the session has been established, only the payloads could be sent to the observer, thus eliminating the overhead due to the inclusion of the CoAP headers in each notification message.

As for service discovery, the CoRE WG has defined a mechanism, denoted as *Resource Directory* (RD) [14], to be adopted in M2M applications. The use of a RD is necessary because of the impracticality of a direct resource discovery, due to the presence of duty-cycled nodes and unstable links in LLNs. Each CoAP server must expose an interface */.well-known/core* to which a client can send requests for discovering available resources. The CoAP server will reply with the list of resources and, for each resource, an attribute that specifies the format of the data associated to that resource. The CoAP protocol, however, does not specify how a node joining the network for the first time must behave in order to announce itself to the resource directory node. In [15], this functionality is extended to multicast communications. In particular, multicast Resource Discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A *GET* request to the appropriate multicast address is made for */.well-known/core*. Of course this multicast Resource Discovery works only within an IP multicast domain and does not scale to larger networks that do not support end-to-end multicast.

The registration of a resource in the RD is performed by sending a POST request to the RD, while the discovery can be accomplished by issuing a GET request to the RD targeting the .well-known/core URI. This discovery mechanism is totally self-contained in CoAP as it uses only CoAP messages.

The adoption of the CoSIP protocol provides an alternative mechanism to register resources on a RD, which may be also called CoSIP Registrar Server. The advantage of using a CoSIP based registration mechanism is that it might be possible to register resources other than those reachable through CoAP, thus providing a scalable and generic mechanism for service discovery in constrained applications with a higher degree of expressiveness, such as setting an expiration time for the registration.

3. SIP. The Session Initiation Protocol (SIP) [10] is an IETF standard application-layer control protocol that can be used to establish, modify, or terminate end-to-end sessions. SIP is a text-based client-server protocol,

where the client sends SIP requests and the server responds to requests. SIP architecture includes both end systems (terminals), also called SIP user agents, and intermediate systems, called SIP proxy, redirect or registrar servers, depending on their function. A "registrar server" is SIP server that receives registration requests issued by SIP user agents, and used for maintaining the binding between the SIP user name also called address-of-record SIP AOR, and its current contact address, that can be used for reaching such user/resource. The mapping between SIP AORs and SIP contact URIs is called Location Service and is an important component for resource discovery in SIP.

All SIP addresses are represented by URIs with the scheme "sip:", and identify a name or contact address of a SIP user; a SIP user can be a real user, a system, an application, or a any kind of resources.

The proxy servers are intermediary entities that act as both server and client for making requests on behalf of other clients. A proxy server may act as "outbound" proxy when used for routing SIP request addressed to a user that is not maintained in a local Location Service, or as "far-end" (or "destination") proxy if the request is addressed to a user with an AOR maintained by the proxy and mapped to one or more SIP contact URIs.

Differently by the proxy servers, the redirect servers accept requests and replies to the client with a response message that typically provides a different contact address (URI) for the target of previous request.

SIP signaling between users consists of requests and responses. When a UA wants to send a request to a remote user (identified by a SIP AOR), it may send the message directly to the IP address of the remote user's UA, or to the proxy server that is responsible for the target AOR (normally the fully qualified domain name (FQDN) of the proxy server is included in the AOR), or to a locally configured outbound proxy server. When the request reaches the target UA, the latter may optionally replies with some provisional 1xx responses and with one final response (codes 2xx for success, 3xx, 4xx, 5xx and 6xx for failure).

SIP defines different request methods such as INVITE, ACK, BYE, CANCEL, OPTIONS, REGISTER, SUBSCRIBE, NOTIFY, atc.

When a UA wants to initiate a session it sends an INVITE message that may be responded with provisional 1xx responses and a final response. The UA that issued the INVITE then have to confirm the final response with a ACK message. Differently by all other SIP transaction, the INVITE transaction is a three-way handshake (INVITE/2xx/ACK).

Once the session is established, both endpoints (user agents) may modify the session with a new INVITE transaction, or tear-down the session with a BYE transaction (BYE/2xx). When the caller or the callee wish to terminate a call, they send a BYE request. SIP messages may contain a "body" that is treated as opaque payload by SIP.

Figure 3.1 shows an example of SIP message flow, including the registration of two UAs with their own registrar/proxy servers, and a session setup and tear-down from UA1 (identified by the SIP AOR sip:u1@P1) to UA2 (identified by the SIP AOR sip:u2@P2).

During an INVITE transaction the SIP body is used to negotiate the session in terms of transport and application protocol, IP addresses and port number, payload formats, encryption algorithms and parameters, etc. The negotiation follows an offer/answer paradigm, where the offer is usually sent within the INVITE while the answer is in the 2xx final response. The most used protocol for such negotiation is the Session description Protocol (SDP); however other prtocol may be used.

4. CoSIP. As described in Section 1, in both constrained and non-constrained environments there are many applications that may require or simply may obtain advantages by negotiating end-to-end data sessions. In this case the communication model consists in a first phase in which one endpoint requests the establishment of a data communication and, optionally, both endpoints negotiate some communication parameters (transfer protocols, data formats, endpoint IP addresses and ports, encryption algorithms and keying materials, and other application specific parameters) of the subsequent data sessions. This may be useful for both client-server or peer-to-peer applications, regardless the data sessions evolve or not according to a request/response model. The main advantage is that all such parameters, including possible resource addressing, may be exchanged in advance, while no such control information is required during data transfer. The longer the data sessions, the more the advantage is evident respect to a per-message control information. Also in the case of data sessions that may vary formats or other parameters during time, such adaptation may be supported by performing session renegotiation. A standard way to achieve all this onto an IP-based network may be by using the Session

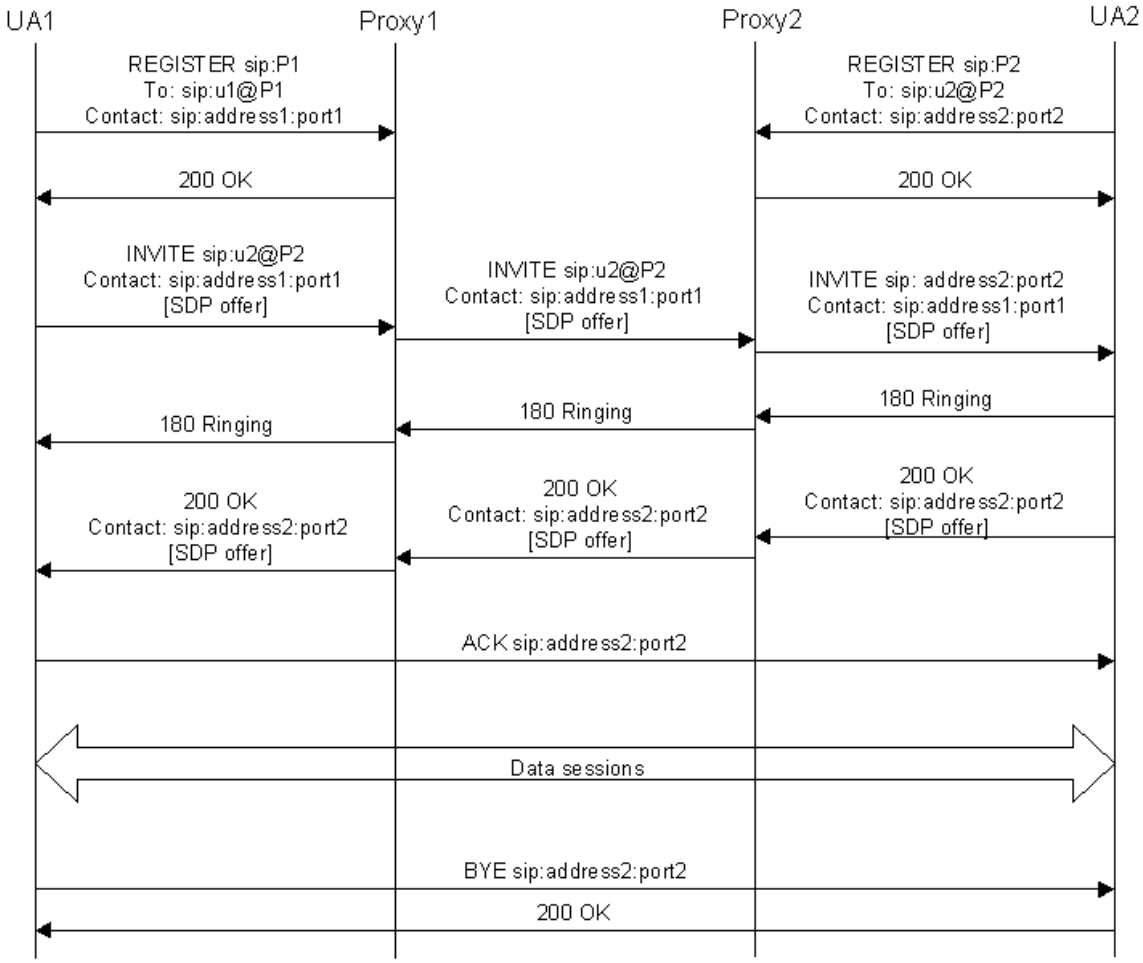

FIG. 3.1. *UA registrations and session setup with two intermediate proxy servers.*

Initiation Protocol [10]. In fact SIP has been defined as standard protocol for initiating, modifying and tearing down any type of end-to-end multimedia sessions. SIP is independent from the protocol used for data transfer and from the protocol used for negotiating the data transfer (such negotiation protocol can be encapsulated transparently within the SIP exchange). In order to simplify the implementation, SIP reuses the same message format and protocol fields of HTTP. However, in contrast to HTTP, SIP works by default onto UDP, by directly implementing all mechanisms for a reliable transaction-based message transfer. This is an advantage in duty-cycled constrained environment where some problems may arise when trying to use connection-oriented transports, such as TCP. However, SIP may also run onto other transport protocols such as TCP, SCTP, TLS or DTLS. Unfortunately SIP derives from HTTP the text-based protocol syntax that, even if it simplifies the implementation and debugging, results in i) larger message sizes, and ii) bigger processing costs and probably larger source code size (RAM footprint) required for message parsing. Note that the SIP standard defines also a mechanism for reducing the overall size of SIP messages; this is achieved by using a compact form of some common header field names. However, although it allows a partial reduction of the message size, it may still result in big messages, especially if compared to other binary formats, for example those defined for CoAP. For this reason we tried to define and implement a new binary format for SIP in order to take advantages of the functionalities already defined and supported by SIP methods and functions, together with a very compact message encoding. We naturally called such new protocol CoSIP, that stands for Constrained Session Initiation Protocol, or, simply, Constrained SIP. Due to the protocol similarities between SIP and HTTP, in order to

maximize the reuse of protocol definitions and source code implementations, we decide to base CoSIP onto the same message format that has been defined for CoAP, thanks to the role that CoAP plays respect to HTTP. However, it is important to note that, while CoAP required to define new message exchanges, mainly due to the fact that CoAP need to operated in constrained and unreliable networked scenario over UDP transport protocol, while HTTP works over TCP, CoSIP may completely reuse all SIP message exchanges and transactions already defined by the SIP standard, since SIP already works over unreliable transport protocols (e.g. UDP).

SIP is structured as a layered protocol, where at the top there is the concept of dialog, that is a peer-to-peer relationship between two SIP nodes that persists for some time and facilitates sequencing of different request-response exchanges (transactions). In CoAP there is no concept equivalent to SIP dialogs, and, if needed, it has to be explicitly implemented at application level. Under the dialog there is the transaction layer, that is the message exchange that comprises a client request, the following optional server provisional responses and the server final response. The concept of transaction is also present in CoAP where requests and responses are bound and matched through a token present as message header field. Under the transaction there is the messaging layer where messages are effectively formatted and sent through an underlying non-SIP transport protocol (such as UDP or TCP). Instead of completely re-designing a session initiation protocol for constrained environments, we propose to reuse the SIP layered architecture of SIP, by simply re-defining the messaging layer with a constrained-oriented binary encoding. For such a purpose, we propose to reuse the same CoAP message syntax [6]. Figure 4.1 shows the CoSIP message format derived by CoAP. A CoSIP message starts with the 2-bit Version field (set to 1, i.e. CoSIP version 1), followed by the 2-bit Type field (set to 1 = Non-confirmable), the 4-bit CoAP TKL field (set to 0), the 8-bit Code field that encode request methods (for request messages) and response codes (for response messages), the 16-bit CoAP Message ID field, followed by zero or more Option fields. In case a CoSIP message body is present, as in CoAP it is appended after Options field, prefixed by an 1-byte marker (0xFF) that separates CoSIP header and payload. Options are encoded as in CoAP in Type-Length-Value (TLV) format and encode all CoSIP header fields (From, Via, Call-ID, etc.) included in the CoSIP message.

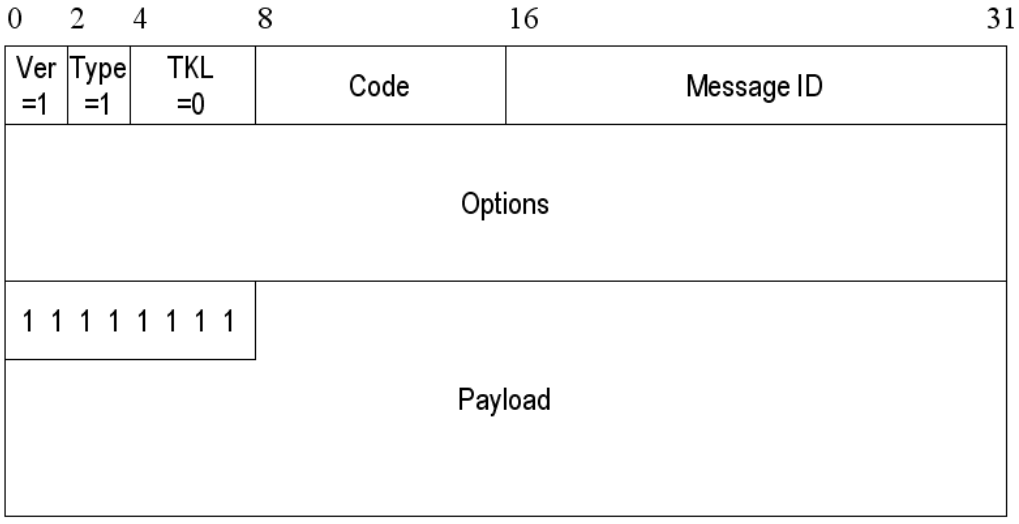

Fig. 4.1. *CoSIP message format.*

Since CoSIP re-uses the transaction layer of SIP, no CoAP optional Token field is needed [6] and the TKL (Token Length) field can be permanently set to 0. Moreover, since CoSIP already has reliable message transmission (within the transaction layer), no Confirmable (0), Acknowledgement (2) nor Reset (3) message types are needed, and the only type of message that must be supported is Non-confirmable (1).

The comparison of the layered architecture of CoSIP and SIP is shown in Fig. 4.2.

Beside the above binary message, a CoSIP message can be virtually seen as a standard SIP message,

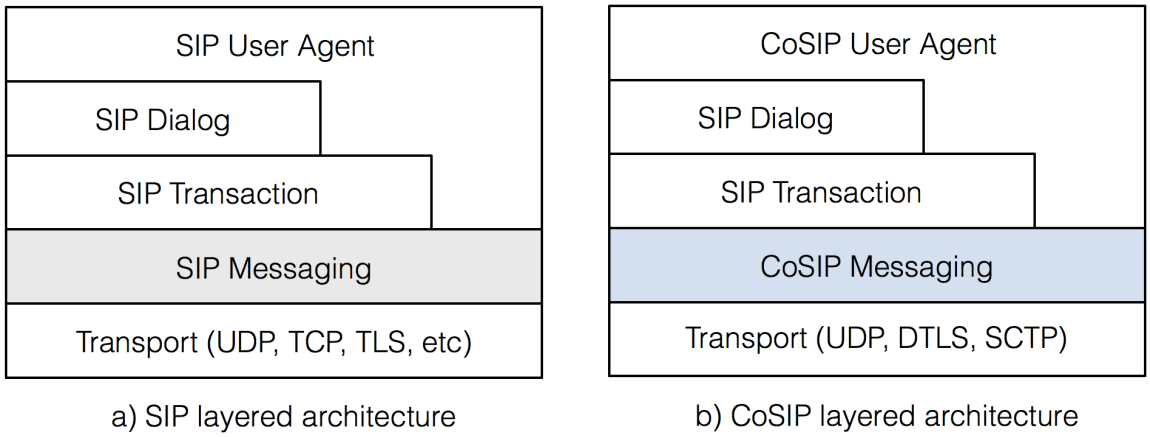

FIG. 4.2. *Comparison of the layered architectures of SIP (a) and CoSIP (b).*

formed by one request–line or one status–line (depending if the message is a request or a response), followed by a sequence of SIP header fields, followed by a message body, if preset. In particular, SIP header fields are logically the same of the standard SIP protocol, properly encoded in corresponding CoSIP Options. For each SIP header field a different option number has been set, and a proper encoding mechanism has been defined. In particular general rules that we followed are:

- IP addresses are encoded as a sequence of 5 bytes for IPv4 and 17 bytes for IPv6, where the first byte discriminates the type of address, i.e. 1 = IPv4 address, 2 = IPv6 address, 3 = FQDN (fully qualified domain name);
- for header field parameters, when possible, the parameter name is implicitly identified by the position of its value in the corresponded binary-encoded CoSIP Option; otherwise parameter names are substituted by parameter codes; in the latter case the parameter is encoded as type-value pair (in case of fixed size values) or type-length-vale tuple (in case of variable size values);
- random tokens, such as SIP "branch" values, SIP "from" and "to" tags, "call-id", etc. are generated as arrays of maximum 6 bytes.

One problem in reusing the current CoAP message format [6] is that in CoAP the 8-bit Code field is used to encode all possible request methods and response codes. In particular, for response messages the 8-bit Code field is divided by CoAP into two sub-fields "class" (3 bits) and "details" (5 bits); the upper three bits ("class") encodes the CoAP response classes 2xx (Success), 4xx (Client Error), and 5 (Server Error), while the remaining 5 bits ("details") encode the sub-type of the response within a given class type. For example a 403 "Forbidden" response is encoded as 4 ("class") and 03 ("details"). Unfortunately, this method limits the number of possible response codes that can be used (for example, using only 5 bits for "details" does not allow the direct encoding of response codes such as 480 "Temporarily Unavailable" or 488 "Not Acceptable Here"). In CoSIP, we overcome this problem by encoding within the Code field only the response class (2xx, 4xx, etc.) and by adding an explicit Option field, called "Response-Code" option, that encodes the complete response code (e.g. 488), including the response sub-type (88, in the case of response code 488). The size of the "Response-Code" option is 2 bytes. Moreover, in order to support all SIP/CoSIP response codes we also added the classes 1xx (Provisional) and 3xx (Redirect) used in SIP.

5. IoT Application Scenarios. In this section, we will describe the most significant for IoT applications, intended to provide an overview of the capabilities and typical usage of the CoSIP protocol. In all the scenarios, we consider a network element, denoted as "IoT Gateway", which includes also a HTTP/CoAP proxy, which can be used by nodes residing outside the constrained network to access CoAP services.

5.1. CoAP Service Discovery. CoSIP allows smart objects to register the services they provide to populate a CoSIP Registrar Server, which serves as a Resource Directory. The terms "Registrar Server" and

"Resource Directory" are here interchangeable.

Figure 5.1 shows a complete service registration and discovery scenario enabled by CoSIP. We consider a smart object that includes a CoAP server, which provides one or more RESTful services, and a CoSIP agent, which is used to interact with the CoSIP Registrar Server. The smart object issues a `REGISTER` request (denoted with the letter "a" in the figure) which includes registration parameters, such as the Address of Record (AoR) of the CoAP service and the actual URL that can be used to access the resource (Contact Address). Note that, while the original SIP specification states that the To header MUST report a SIP or SIPS URI, CoSIP allows to specify any scheme URI in the To header, e.g. a CoAP URI. Upon receiving the registration request, the Registrar Server stores the AoR-to-Contact Address mapping in a Location Database and then sends a `200 OK` response.

When a REST client, either CoAP or HTTP, is willing to discover the services, it can issue a `GET` request targeting the .well-known/core URI, which is used as a default entry point to retrieve the resources hosted by the Resource Directory, as defined in [16]. The `GET` request is sent to the HTTP/CoAP proxy, which returns a `200 OK` (in the case of HTTP) or a `2.05 Content` (in the case of CoAP) response containing the list of services in the payload.

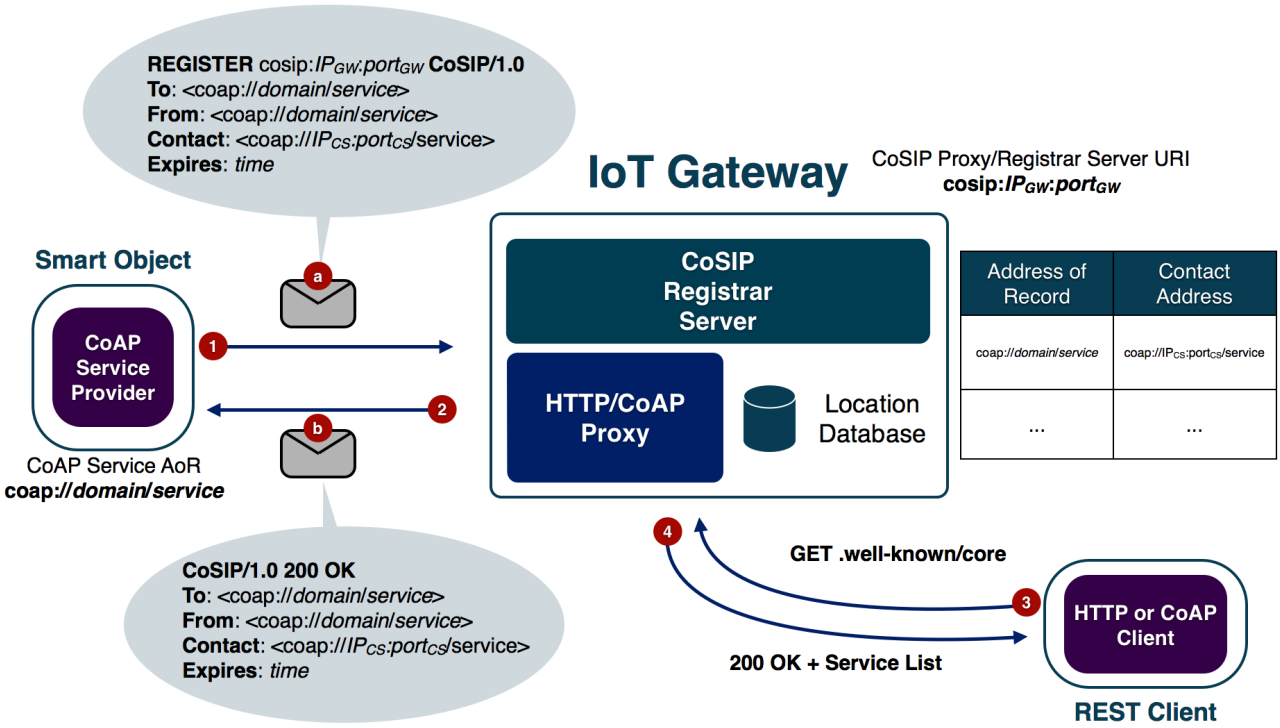

FIG. 5.1. *CoAP Service Discovery.*

5.2. Session Establishment. A session is established when two endpoints need to exchange data. CoSIP allows the establishment of session in a standard way without binding the session establishment method to a specific session protocol. For instance, CoSIP can be used to negotiate and instantiate a RTP session between constrained nodes. Once a session has been established, the data exchange between the endpoints occurs (logically) in a peer-to-peer fashion.

Figure 5.2 shows how CoSIP can be used to establish a session between two endpoints. Let's assume an IoT Agent (IoT-A_1) identified by the CoSIP URI *cosip:user1@domain*, which includes at least a CoSIP agent, has registered its contact address to an IoT Gateway in the same way as described in the previous subsection (steps 1 and 2). If another IoT-A_2 *cosip:user2@domain* wants to establish a session with IoT-A_1, it will send a proper `INVITE` request to the IoT Gateway, which will act as a CoSIP Proxy relaying the request to IoT-A_1

(steps 3 and 4). IoT-A_1 will then send a `200 OK` response to IoT-A_2 (steps 5 and 6), which will finalize the session creation by sending an `ACK` message to IoT-A_2 (steps 7 and 8).

At this point the session has been setup and data flow between IoT-A_1 and IoT-A_2 can occur directly. The session establishment process can be used to negotiate some communication parameters, for instance by encapsulating Session Description Protocol (SDP) [12] or equivalent in the message payload. As we will show in the evaluation section, setting up a session, rather than using CoAP, both in a request/response or subscribe/notify paradigm, is a very efficient approach to avoid the burden of the overhead due to carrying headers in each exchanged message since eventually only the payloads would be relevant for the application.

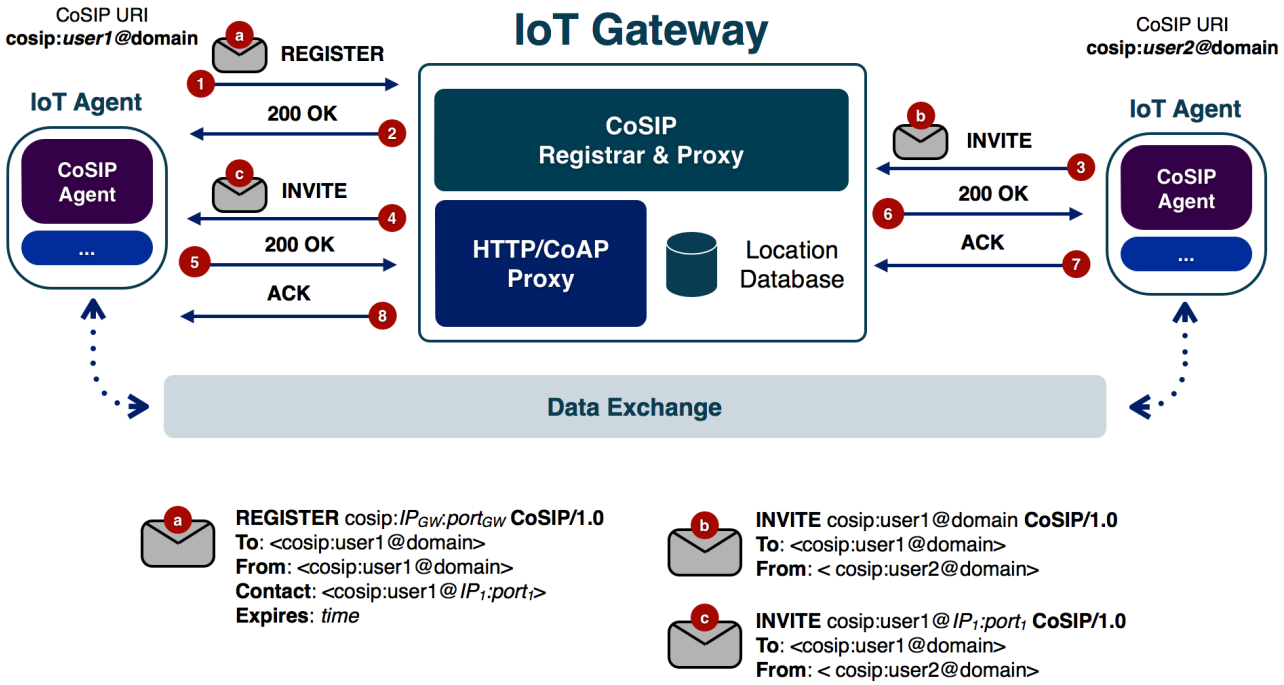

FIG. 5.2. *CoSIP Session Establishment.*

5.3. Subscribe/Notify Applications. IoT scenarios typically involve smart objects which might be battery-powered devices. It is crucial to adopt energy-efficient paradigms, e.g. OS tasks, application processing, and communication. In order to minimize the power consumed, duty-cycled smart objects are adopted. Sleepy nodes, especially those operating in LLNs, aren't guaranteed to be reached, therefore it is more appropriate for smart objects to use a Subscribe/Notify, also denoted as Publish/Subscribe (Pub/Sub), approach to send notifications regarding the state of their resources, rather than receive and serve incoming requests. Such a behavior can be achieved by leveraging on the inherent capabilities of SIP, and therefore of CoSIP, as sketched in Fig. 5.3.

The depicted scenarios considers several Pub/Sub interactions: notifications can be sent either by a Notifier IoT Agent (IoT-A_N) or by an IoT Gateway, and subscribers can be either Subscriber IoT Agents (IoT-A_S), IoT Gateways, or generic Remote Subscribers. Let's assume that all the notifiers have previously registered with their CoSIP Registrar Server (this step is also denoted as the Publishing phase in a typical Pub/Sub scenario). The standard subscription/notification procedure is the following:

1. the subscriber sends a `SUBSCRIBE` request to the notifier, also specifying the service events it is interested in;
2. the notifier stores the subscriber's URI and event information and sends a `200 OK` response to the subscriber;
3. whenever the notifier's state changes, it sends a `NOTIFY` request to the subscriber;

4. the subscriber sends a `200 OK` response back to the notifier.

Figure 5.3 reports all the use cases when a Pub/Sub might be used. An IoT-A_S can subscribe to the service of an IoT-A_N in the same network, in case it is willing to perform some task, such as data/service aggregation. The IoT Gateway can subscribe to the IoT-A_N's in order to collect sensed data, e.g. to store them in the cloud, without the need to periodically poll for data. Finally, the IoT Gateway itself might be a notifier for remote subscribers, which are interested in notifications for specific services provided by the gateway, which may or may not be the same of existing IoT-A_N nodes managed by the gateway. Note that, it might be possible to have interaction with legacy SIP agents in case the IoT Gateway is also able to perform SIP/CoSIP proxying.

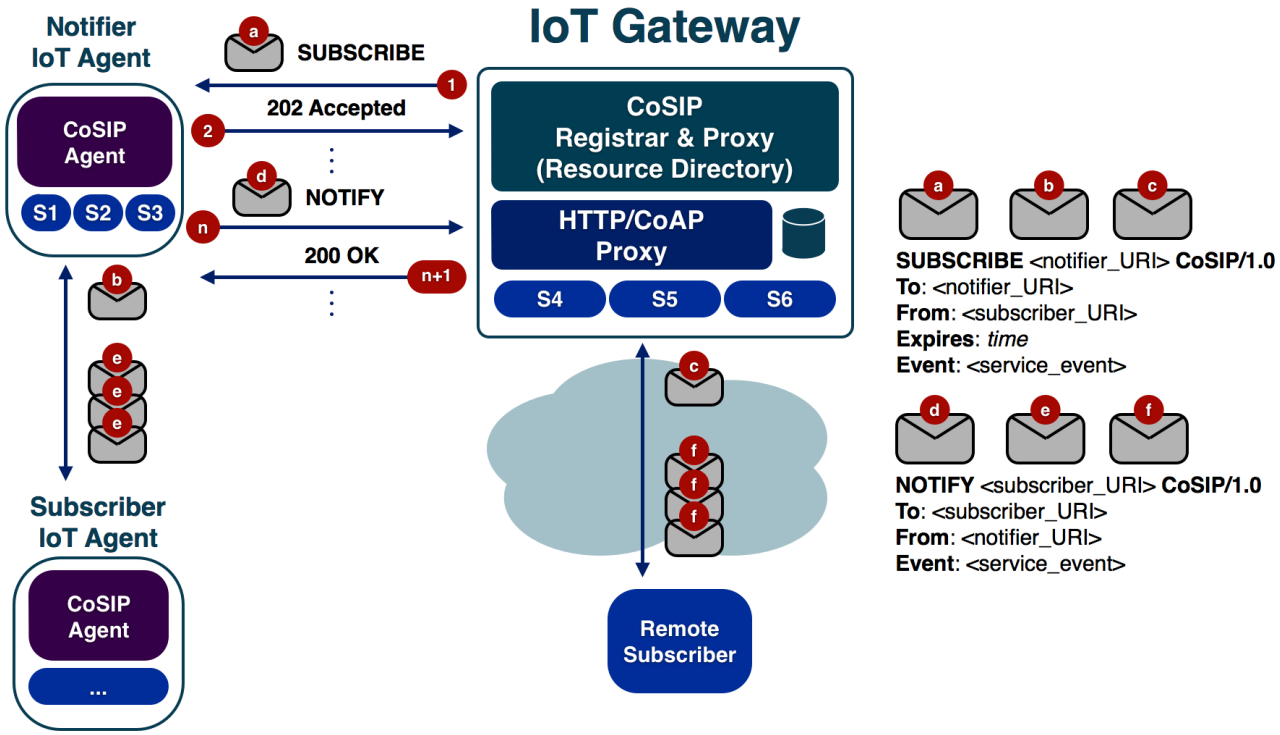

FIG. 5.3. *Subscribe/Notify applications with CoSIP.*

The adoption of CoSIP in IoT scenarios allows to easily set up efficient Pub/Sub-based applications in a standard way, thus allowing for seamless integration and interaction with the Internet. Moreover, the valuable experience gained in the past years with SIP, both in terms of technologies and implementations, can be reused to speed up the implementation and deployment of session-based applications.

6. Protocol Evaluation. In order to evaluate the performance of CoSIP, an implementation of the protocol has been developed together with some test applications. In this work, we have decided to focus on network performance as a metric by measuring the amount of network traffic generated by the test applications. The CoSIP protocol has been implemented in Java language, due to its simplicity, cross-platform support, and the availability of already developed SIP and CoAP libraries [17, 18]. The source code of the CoSIP implementation is freely available at [19].

The performance results show that many advantages can be achieved by using CoSIP, both in constrained and non-constrained applications. The first evaluation compares CoSIP and SIP in terms of bytes transmitted for the signaling part related to the instantiation and termination of a session. Each CoSIP request and response message is separately compared with its SIP counterpart. The results are illustrated in Fig. 6.1. Table 6.1 shows the compression ratio for each CoSIP/SIP message pair. Regarding the session as a whole, CoSIP yields an overall compression ratio of slightly more than 0.55.

Message type	CoSIP (bytes)	SIP (bytes)	compression ratio
INVITE	311	579	0.537
100 Trying	141	279	0.505
180 Ringing	173	372	0.465
200 OK	293	508	0.577
ACK	216	363	0.595
BYE	183	309	0.592
200 OK	162	274	0.591

TABLE 6.1
Comparison between CoSIP and SIP signaling (bytes per message) for session instantiation and establishment.

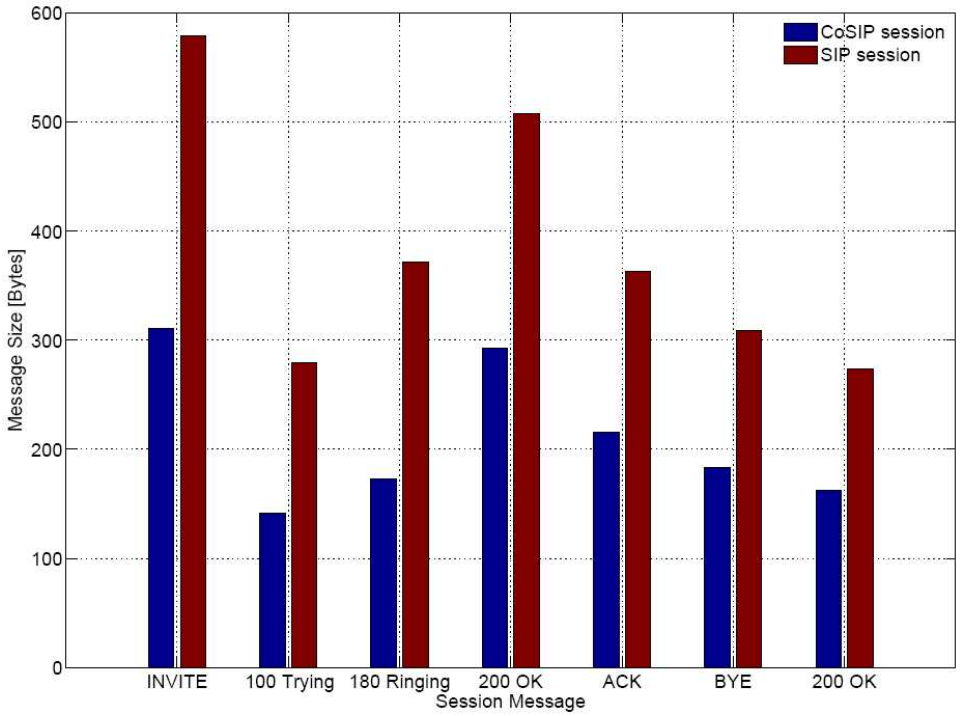

FIG. 6.1. *Transmitted bytes for CoSIP and SIP session (signaling only).*

Another evaluation has been made to show the advantage of using session in constrained applications. Figure 6.2 shows the amount of network traffic (in bytes) generated by two constrained applications: the first application uses CoSIP to establish a session and then performs the data exchange by sending the payloads over UDP; the second is a standard CoAP-based application where the communication occurs between a CoAP client and a CoAP server, using confirmed CoAP POST requests. In both cases data is sent at the same rate of one data message every 2 seconds. The figure shows that the lightweight CoSIP session is instantiated in a very short period of time and after the session has been established few bytes are exchanged between the endpoints. On the other hand the CoAP-based application has no overhead at the beginning due to the instantiation of the session but, soon after, the amount of traffic generated by this application exceeds that of the CoSIP-based application, since in the CoAP-based scenario data is exchanged within CoAP messages resulting in an unnecessary CoAP overhead.

Note that in the depicted scenario the CoSIP signaling used for session initiation includes all SIP header fields normally used in standard non-constrained SIP application, that is no reduction in term of header fields has been performed. Instead for the CoAP application we considered only mandatory CoAP header fields

resulting in the best-case scenario for CoAP in term of CoAP overhead (minimum overhead). This means that in other CoAP applications the slope of the line could become even steeper, thus reducing the time when the break-even point with CoSIP is reached.

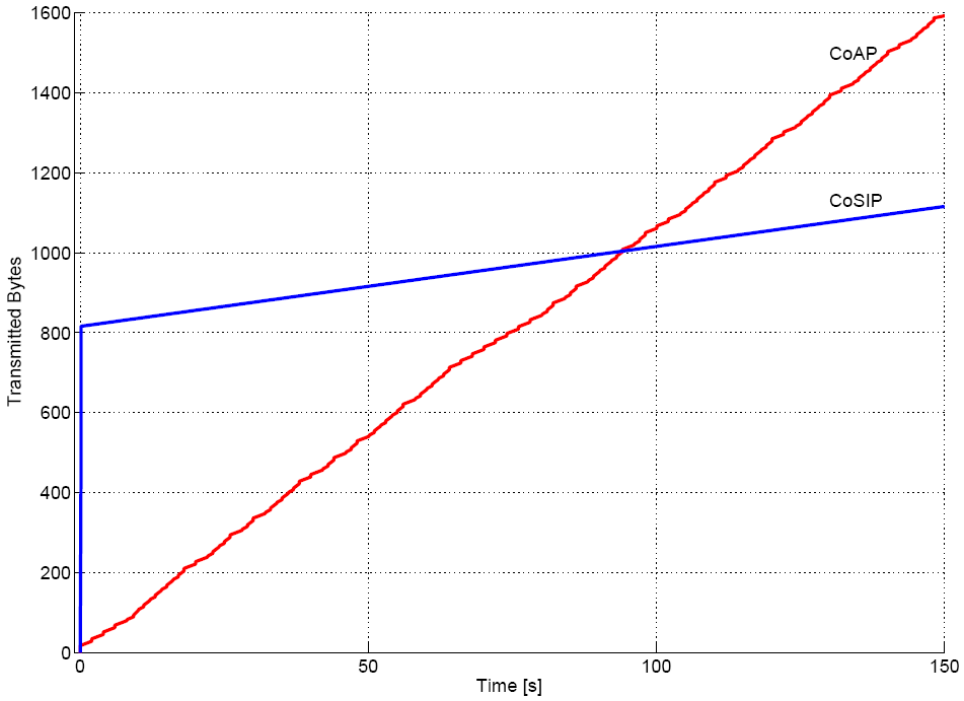

FIG. 6.2. *Transmitted bytes in a CoSIP Session vs. CoAP confirmed POST requests and responses.*

7. Conclusions. In this paper, we have introduced a low-power protocol, named "CoSIP", for establishing sessions between two or more endpoints targeting constrained environments. Many applications, both in constrained and non-constrained scenarios, do benefit from establishing a session between the participants in order to minimize the communication overhead and to negotiate some parameters related to the data exchange that will occur. The CoSIP protocol is a constrained version of the SIP protocol intended to minimize the amount of network traffic, and therefore energy consumption, targeted for IoT scenarios. A similar effort in trying to minimize the amount of data in IoT and M2M applications is being carried on in standardization organizations, such as the IETF CoRE Working Group, which is currently defining a protocol (CoAP) to be used as a generic web protocol for RESTful constrained environments and maps to HTTP. Similarly, in this work we have proposed to apply the same approach to define a protocol for session instantiation, negotiation, and termination. We have described some interesting IoT scenarios that might benefit from using such a protocol, namely service discovery, session establishment, and services based on a subscribe/notify paradigm. A Java-language implementation of CoSIP has been developed and tested to evaluate the performance of the newly proposed protocol, by measuring the amount of transmitted bytes compared to other solutions based on SIP and CoAP respectively. The results show that applications that use CoSIP can outperform other SIP- and CoAP-based applications in terms of generated network traffic: SIP signaling can be compressed of nearly 50% using CoSIP, and long-running applications that may use CoAP for sending the same type of data to a given receiver may be better implemented with CoSIP, since no CoAP overhead has to be transmitted along with each transmitted data message leading to a packet size and per-packet processing reduction; packet size reduction in turn may reduce the need for packet fragmentation (in 6LoWPAN networks) and the energy consumption of the nodes involved in the data exchange.

Future work will include an exhaustive experimentation, both in simulation environments and a real-world testbed comprising a variety of heterogeneous devices which is currently being setup at the Department of

Information Engineering of the University of Parma, aimed to evaluate the performance of the CoSIP protocol both in terms of energy consumption and delay. The tests will focus on the time required to setup a session in different scenarios, such as in IEEE 802.15.4 multi-hop environments, and the measurement of energy consumption with a comparison between CoSIP sessions and standard CoAP communication. Two different perspectives will be analyzed: i) end-to-end delay between the actual session participants and ii) energy consumption on intermediate nodes which will be indirectly involved in the session as responsible for multi-hopping routing at lower layers. The target platforms will be both constrained and non-constrained devices for session participants and relay nodes, in order to provide a thorough evaluation comprising heterogeneous devices operating under different conditions.

REFERENCES

[1] C. Bormann, *Guidance for Light-Weight Implementations of the Internet Protocol Suite*, IETF Internet-Draft *draft-ietf-lwig-guidance* (February 2013), `http://tools.ietf.org/id/draft-ietf-lwig-guidance`

[2] S. Deering, and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force, RFC 2460 (December 1998).

[3] IETF IPv6 over Low Power WPAN Working Group. `http://tools.ietf.org/wg/6lowpan/`

[4] IETF Routing Over Low power and Lossy networks Working Group. `http://tools.ietf.org/wg/roll/`

[5] IETF Constrained RESTful Environments Working Group. `http://tools.ietf.org/wg/core/`

[6] Z. Shelby, K. Hartke, K., and C. Bormann, *Constrained Application Protocol (CoAP)*, IETF Internet-Draft *draft-ietf-core-coap* (May 2013), `http://tools.ietf.org/id/draft-ietf-core-coap`

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, Internet Engineering Task Force, RFC 2616 (June 1999).

[8] J. Postel, *User Datagram Protocol*, Internet Engineering Task Force, RFC 768 (August 1980).

[9] K. Hartke, *Observing Resources in CoAP*, IETF Internet-Draft *draft-ietf-core-observe* (February 2013), `http://tools.ietf.org/id/draft-ietf-core-observe`

[10] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *SIP: Session Initiation Protocol*, Internet Engineering Task Force, RFC 3261 (June 2002).

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Internet Engineering Task Force, RFC 3550 (July 2003).

[12] V. Jacobson, and C. Perkins, *SDP: Session Description Protocol*, Internet Engineering Task Force, RFC 4566 (July 2006).

[13] A. B. Roach, *Session Initiation Protocol (SIP)-Specific Event Notification*, Internet Engineering Task Force, RFC 3265 (June 2002).

[14] Z. Shelby, S. Krco, and C. Bormann, *CoRE Resource Directory*, IETF Internet-Draft *draft-ietf-core-resource-directory* (June 2013), `http://tools.ietf.org/id/draft-ietf-core-resource-directory`

[15] A. Rahman and E. Dijk, *Group Communication for CoAP*, IETF Internet-Draft *draft-ietf-core-groupcomm* (July 2013), `http://tools.ietf.org/id/draft-ietf-core-groupcomm`

[16] Z. Shelby, *Constrained RESTful Environments (CoRE) Link Format*, Internet Engineering Task Force, RFC 6690 (August 2012).

[17] mjSIP project. `http://mjsip.org/`

[18] mjCoAP project. `http://mjcoap.org/`

[19] CoSIP project. `http://cosip.org/download/`

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014