CoSIP: a Constrained Session Initiation Protocol for the Internet of Things

Simone Cirani, Marco Picone, and Luca Veltri

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

Abstract. The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects" (SO), in an Internet-like structure. SOs typically feature limited capabilities in terms of computation and memory and operate in constrained environments, such low-power lossy networks. As IP has been foreseen as the standard for smart-object communication, an effort to bring IP connectivity to SOs and define suitable communication protocols (i.e. CoAP) is being carried out within standardization organisms, such as 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.

Keywords: Internet of Things, service discovery, communication protocols, constrained applications, SIP, CoAP

1 Introduction

The Internet of Things (IoT) refers to the interconnection of billions of constrained devices, denoted as "smart objects" (SO), 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, it is required to use standard and interoperable communication mechanisms. The use of IP has been foreseen as the standard for interoperability for smart objects by standardization organisms, such as the IETF and the IPSO Alliance. As billions of smart objects are expected to come to life and IPv4 addresses have eventually reached depletion, IPv6 [1] has been identified as a candidate for smart-object communication. Within the IETF, several working groups have been set in order to address the issues related to smart-object communication. The IETF 6LoWPAN Working Group [2] 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 application layer, the IETF CoRE Working Group [3] is currently defining a Constrained Application Protocol (CoAP) [4], to be used as a generic web protocol for RESTful constrained environments, targeting Machine-to-Machine (M2M) applications, and that can be seen as a compressed version of HTTP [5]. CoAP includes 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 and Internet media types.

The typical Internet of Things protocol stack, compared with the standard web protocol stack, is depicted in Figure 1. As CoAP is designed to be used by constrained nodes, such as duty-cycled devices, in constrained environments, CoAP uses UDP [6] as a lightweight transport. Besides a request/response communication paradigm, CoAP also supports a publish/subscribe paradigm, by letting CoAP clients to "observe" resources. When observing resources, the CoAP server, following client's subscription, sends multiple responses to the client.

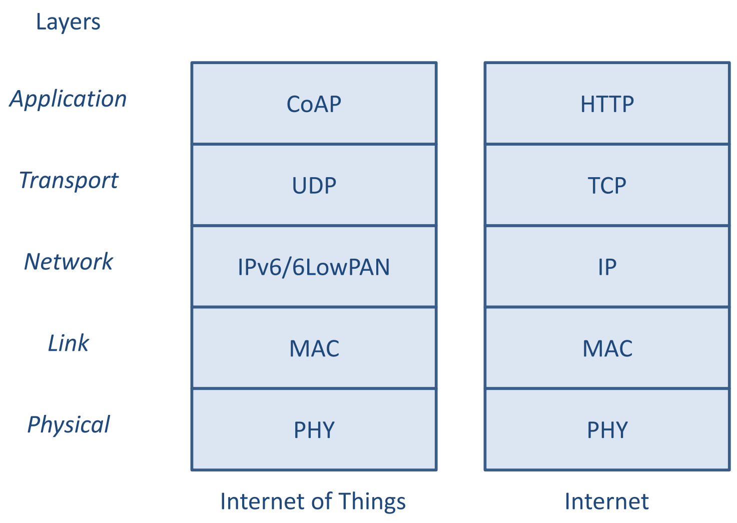


Fig. 1. Comparison between the IoT and the Internet protocol stack for OSI layers.

There are many other applications in both constrained and non-constrained environments that feature non-request/response communication model. Some of these applications require the creation and management of a session. A session is an exchange of data between an association of participants.

The Session Initiation Protocol (SIP) [7] is an Internet application-layer control protocol which aims at enabling the endpoints of the communication to create, modify, and terminate multimedia sessions, such as VoIP calls, multimedia conferences, or any point-to-multipoint data distribution communications. Once a multimedia session has been established, the media are transmitted typically by using other application-layer protocols, such as RTP and RTCP [8], 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 [9]. 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.

As asserted before, also in constrained environments, establishing a session is a likely event to occur, as communication between nodes might go beyond simple request/response pairs, as accomplished by using CoAP. 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, it could be much more efficient to instantiate a session between constrained nodes first, while negotiating some parameters for subsequent communication, and then perform a more lightweight communication by using the session parameters already negotiated, rather then having to carry on the burden of CoAP headers, which in most cases would end up being equal in any request/response.

Such session initiation could be still performed through the standard SIP protocol, also in constrained environments. Moreover, SIP and its extensions, such as event notification [10], already include mechanisms that are being defined for CoAP, such as observing resources [11], which allows for a subscribe/notify communication paradigm, and resource directory [12], which can be used for resource discovery. For this reason, SIP appears to be a suitable alternative to many mechanisms defined in CoAP and related proposals that might be used to address these issues.

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. CoSIP is a binary protocol which maps to SIP, similarly to CoAP and 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, the CoSIP protocol is detailed together with its architecture and preliminary implementation. Use cases for CoSIP-based applications in Internet of Things scenarios are presented in Section 4 and a performance evaluation of the proposed protocol is shown in Section 5. Finally, in Section 6 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 [2] 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 [13] was chartered to develop optimal IPv6 routing in LLNs. Finally, the CoRE (Constrained RESTful Environments) WG [3] 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 [11] allows to let CoAP clients 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) [12], 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. 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 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 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 Initiation Protocol [7]. 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, differently from 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 larger message sizes and bigger processing costs (and 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 message format for SIP in order to take advantages of the functionalities already defined and supported by SIP and by a new binary and 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 [4]. Figure 2 shows the CoSIP message format derived by CoAP. A CoSIP message starts with the 2-bit Version field (1), followed by the 2-bit Type field (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. For each header field a different option number has been set. In order to increase the SIP-to-CoSIP compression ratio, alternatively of encoding the header field value as an opaque byte string, a per SIP header field encoding rule has been also defined.

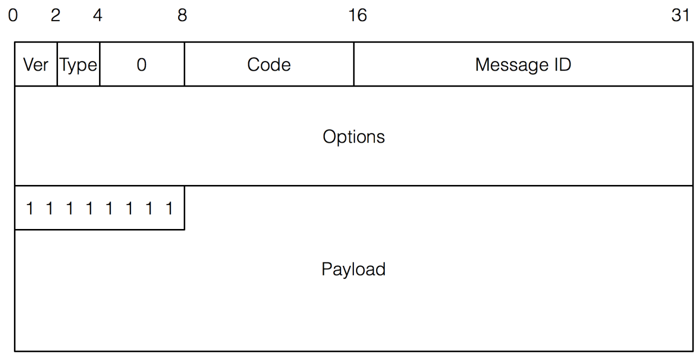


Fig. 2. CoSIP message format.

Since CoSIP re-uses the transaction layer of SIP, no CoAP optional Token field is needed [4] 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 Figure 3.

One problem in reusing the current CoAP message format [4] 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

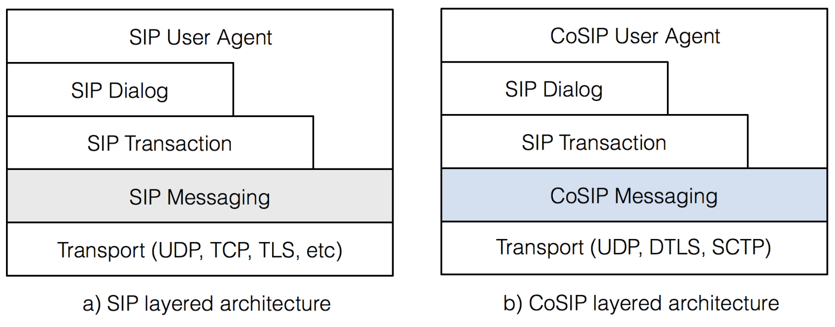

SIP User Agent
SIP Dialog
SIP Transaction
SIP Messaging
Transport (UDP, TCP, TLS, etc)
a) SIP layered architecture
CoSIP User Agent
SIP Dialog
SIP Transaction
CoSIP Messaging
Transport (UDP, DTLS, SCTP)
b) CoSIP layered architecture

Fig. 3. Comparison of the layered architectures of SIP (a) and CoSIP (b).

that encodes the response sub-type. Moreover, in order to support all SIP/CoSIP response codes we also added the classes 1xx (Provisional) and 3xx (Redirect) used in SIP.

4 IoT Application Scenarios

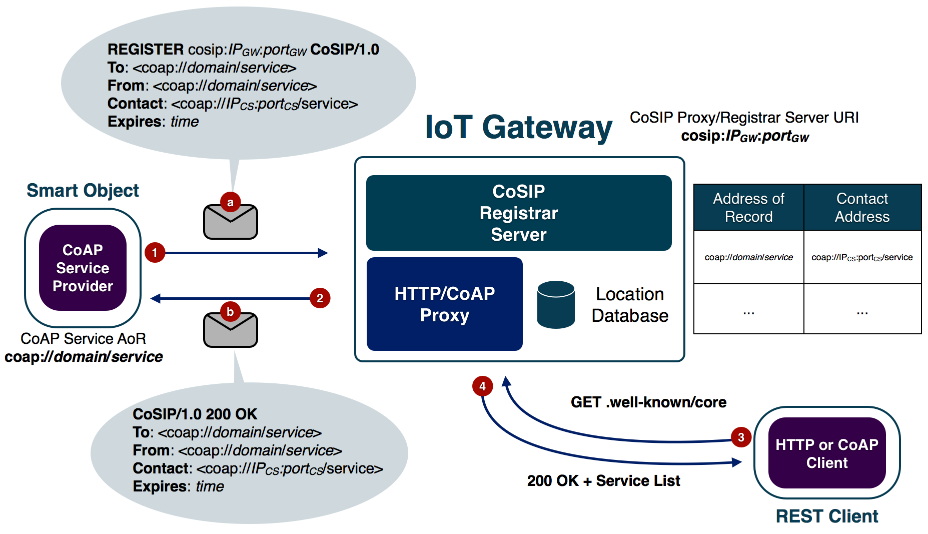
In this section, we will describe the most significant for IoT applications, intended to provide an overview of the capabilities and typical usage of the CoSIP protocol. In all the scenarios, we consider a network element, denoted as "IoT Gateway", which includes also a HTTP/CoAP proxy, which can be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

4.1 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 4 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 [14]. The `GET` request is sent to the HTTP/CoAP proxy, which returns a `200 OK` (in case of HTTP) or a `2.05 Content` (in case of CoAP) response containing the list of services in the payload.



```
REGISTER cosip:IPGW:portGW CoSIP/1.0
To: <coap://domain/service>
From: <coap://domain/service>
Contact: <coap://IPCS:portCS/service>
Expires: time
```

```
CoSIP/1.0 200 OK
To: <coap://domain/service>
From: <coap://domain/service>
Contact: <coap://IPCS:portCS/service>
Expires: time
```

Address of Record	Contact Address
coap://*domain*/*service*	coap://IP$_{CS}$:port$_{CS}$/service
...	...

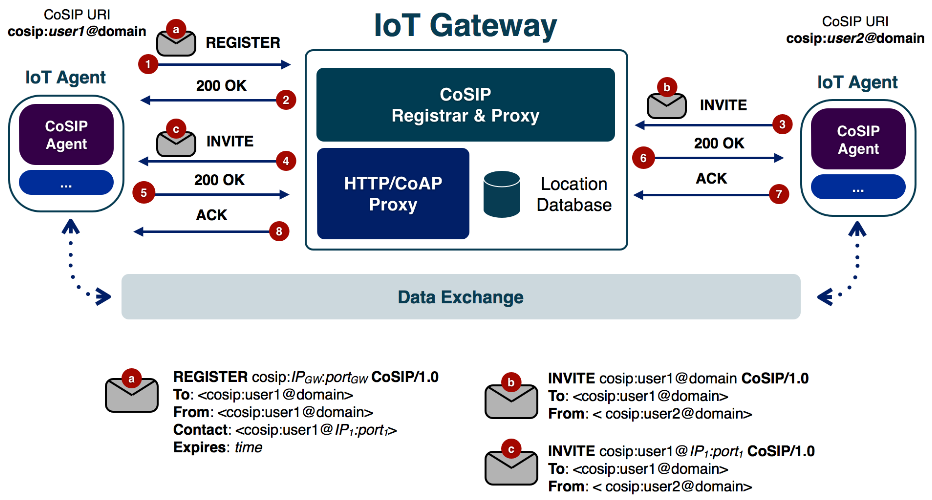
Fig. 4. CoAP Service Discovery.

4.2 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 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) [9] 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.



REGISTER cosip:IP_{GW}:$port_{GW}$ **CoSIP/1.0**
To: <cosip:user1@domain>
From: <cosip:user1@domain>
Contact: <cosip:user1@IP_1:$port_1$>
Expires: *time*

INVITE cosip:user1@domain **CoSIP/1.0**
To: <cosip:user1@domain>
From: < cosip:user2@domain>

INVITE cosip:user1@IP_1:$port_1$ **CoSIP/1.0**
To: <cosip:user1@domain>
From: < cosip:user2@domain>

Fig. 5. CoSIP Session Establishment.

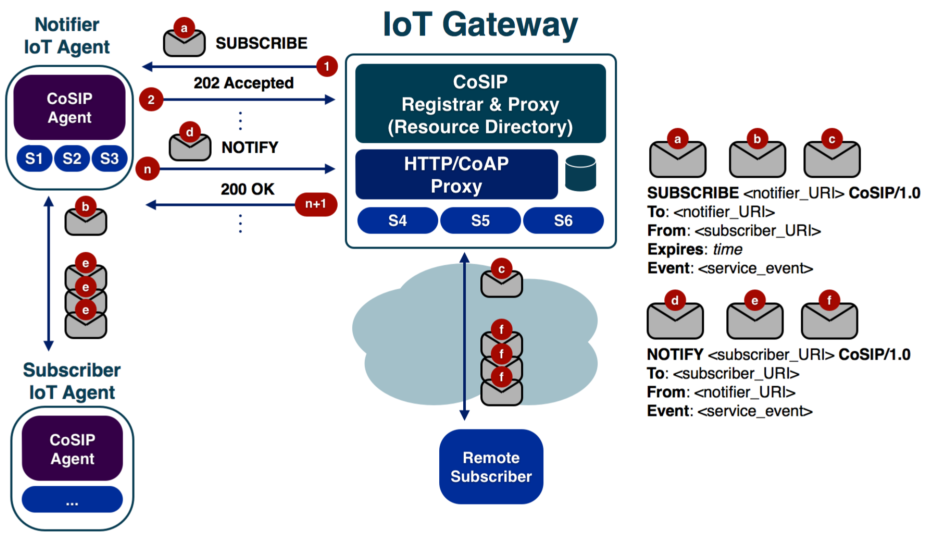
4.3 Subscribe/Notify Applications

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 Figure 6.

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 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 6 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.



SUBSCRIBE <notifier_URI> **CoSIP/1.0**
To: <notifier_URI>
From: <subscriber_URI>
Expires: *time*
Event: <service_event>

NOTIFY <subscriber_URI> **CoSIP/1.0**
To: <subscriber_URI>
From: <notifier_URI>
Event: <service_event>

Fig. 6. 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.

5 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 [15, 16]. The source code of the CoSIP implementation is freely available at [17].

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 signalling 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 Figure 7. Table 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 1. Comparison between CoSIP and SIP signalling (bytes per message) for session instantiation and establishment.

Another evaluation has been made to show the advantage of using session in constrained applications. Figure 8 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

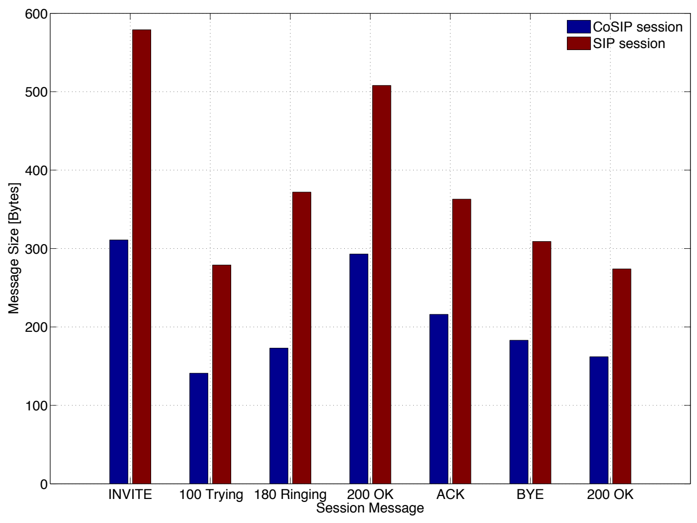


Fig. 7. Transmitted bytes for CoSIP and SIP session (signalling only).

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.

6 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

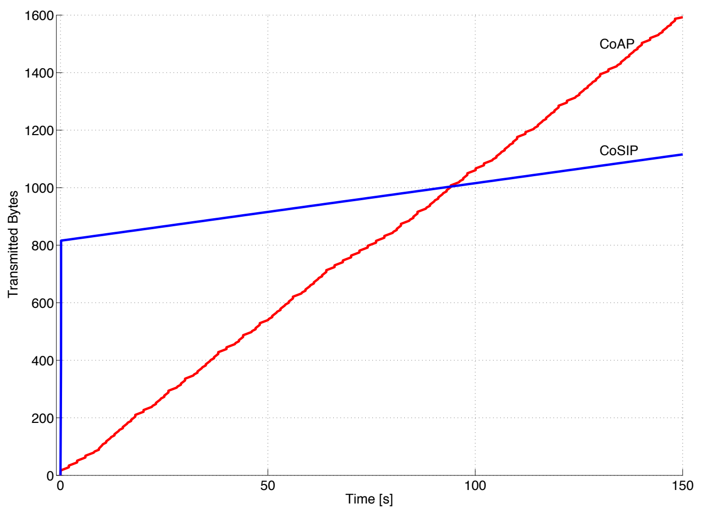


Fig. 8. Transmitted bytes in a CoSIP Session vs. CoAP confirmed POST requests and responses.

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 signalling can be compressed of nearly 50% and long-running applications based on CoAP require less bytes to be transmitted since CoAP options do not need to be sent along in each transmitted message, thus reducing the need for packet fragmentation (in 6LoWPAN networks) and the energy consumption of the nodes involved in the data exchange.

Acknowledgments.

The work of Simone Cirani and Luca Veltri is 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.

References

1. Deering, S., Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard) (December 1998) Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.
2. IETF IPv6 over Low Power WPAN Working Group. `http://tools.ietf.org/wg/6lowpan/`
3. IETF Constrained RESTful Environments Working Group. `http://tools.ietf.org/wg/core/`
4. Shelby, Z., Hartke, K., Bormann, C.: Constrained Application Protocol (CoAP). IETF Internet-Draft *draft-ietf-core-coap* (May 2013) `http://tools.ietf.org/id/draft-ietf-core-coap`
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (June 1999) Updated by RFCs 2817, 5785, 6266, 6585.
6. Postel, J.: User Datagram Protocol. RFC 768 (INTERNET STANDARD) (August 1980)
7. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard) (June 2002) Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878.
8. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD) (July 2003) Updated by RFCs 5506, 5761, 6051, 6222.
9. Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol. RFC 4566 (Proposed Standard) (July 2006)
10. Roach, A.B.: Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265 (Proposed Standard) (June 2002) Obsoleted by RFC 6665, updated by RFCs 5367, 5727, 6446.
11. Hartke, K.: Observing Resources in CoAP. IETF Internet-Draft *draft-ietf-core-observe* (February 2013) `http://tools.ietf.org/id/draft-ietf-core-observe`
12. Shelby, Z., Krco, S., Bormann, C.: CoRE Resource Directory. IETF Internet-Draft *draft-ietf-core-resource-directory* (June 2013) `http://tools.ietf.org/id/draft-ietf-core-resource-directory`
13. IETF Routing over Low Power and Lossy Networks Working Group. `http://tools.ietf.org/wg/roll/`
14. Shelby, Z.: Constrained RESTful Environments (CoRE) Link Format. RFC 6690 (Proposed Standard) (August 2012)
15. mjSIP project. `http://mjsip.org/`
16. mjCoAP project. `http://mjcoap.org/`
17. CoSIP project. `http://cosip.org/download/`