Integrating Wireless Sensor Networks with the Web

W. Colitti, K. Steenhaut and N. De Caro
Information Processing in Sensor Networks Conference 2011

Presenter - Bob Kinicki
Outline

- Introduction
  - REST
- CoAP
  - Request/response Layer
  - Transaction Layer
- CoAP versus HTTP Power Consumption Evaluation
- Integrating CoAP-based WSN with HTTP-based Web Application
- Conclusions and Critique
Introduction

• This paper is highly cited because it discusses an early Contiki implementation of the Constrained Application Protocol (CoAP) on Tmote Sky sensor motes.

• REpresentation State Transfer (REST) identifies a resource (an object) controlled by the server by a URI (Universal Resource Identifier). {Note - the sensor is viewed as the server in this abstraction.}

• Majority of REST architectures use HTTP with its commands: GET, PUT, POST and DELETE.
IETF Constrained RESTful environments (CoRE) Working Group standardized the web service paradigm into networks of smart objects.

In the Web of Things (WoT), object applications are built on top of the REST architecture.

The CoRE group defined a REST-based web transfer protocol called Constrained Application Protocol (CoAP).
CoAP

- **CoAP** manipulates Web resources using the same methods as **HTTP**: GET, PUT, POST and DELETE.

- **CoAP** is a subset of **HTTP** functionality re-designed for low power embedded devices such as sensors (for IoT and M2M).

- **CoAP's** two layers are:
  - Request/Response Layer
  - Transaction Layer
TCP overhead is too high and its flow control is not appropriate for short-lived transactions.

UDP has lower overhead and supports multicast.

Called messaging layer in previous paper.
CoAP

- **Request/Response layer** :: is responsible for transmission of requests and responses. This is where **REST**-based communication occurs.
  - **REST request** is piggybacked on **Confirmable** or **Non-confirmable** message.
  - **REST response** is piggybacked on the related **Acknowledgement** message.

- **CoAP** uses **tokens** to match request/response in asynchronous communications.
**CoAP**

- **Transaction layer**: handles single message exchange between end points.

- **Four message types**:
  - **Confirmable** – requires an **ACK**.
  - **Non-confirmed** – no **ACK** needed.
  - **Acknowledgement** – **ACKs** a **Confirmable**.
  - **Reset** – indicates a **Confirmable** message has been received but context is missing for processing.
CoAP

- **CoAP** provides reliability **without** using TCP as transport protocol.

- **CoAP** enables asynchronous communication.
  - e.g., when CoAP server receives a request which it cannot handle immediately, it first **ACKs** the reception of the message and sends back the response in an off-line fashion. {Not implemented in this study!}

- The transaction layer also supports multicast and congestion control.
CoAP design goals: small message overhead and limited fragmentation.

CoAP uses compact fixed-length 4-byte binary header followed by compact binary options.

Typical request with all encapsulation has a 10-20 byte header.

CoAP implements an observation relationship whereby an “observer” client registers itself using a modified GET to the server.

When resource (object) changes state, server notifies the observer.
CoAP server implemented on Tmote Sky sensor motes running Contiki with 6LowPAN/RPL.
- Asynchronous transactions, observations and congestion control were missing!

HTTP server implemented using same motes.

In experiments, client requests temperature and humidity from server every 10 secs. for 20 minutes.
Power Consumption Tests

- Both CoAP and HTTP servers respond using JSON (lightweight text standard) and not XML.

- Example response from server:

  ```json
  {
  "sensor":"0212:7400:0002:0202",
  "readings":{"hum":31,"temp":23.1}
  }
  ```

  Lower bytes of IP address identifies the sensor mote.
Table 1: CoAP vs HTTP Power Usage

Table 1. Comparison between CoAP and HTTP

<table>
<thead>
<tr>
<th></th>
<th>Bytes per-transaction</th>
<th>Power</th>
<th>Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoAP</td>
<td>154</td>
<td>0.744 mW</td>
<td>151 days</td>
</tr>
<tr>
<td>HTTP</td>
<td>1451</td>
<td>1.333 mW</td>
<td>84 days</td>
</tr>
</tbody>
</table>

- **HTTP** transaction bytes are 10 times higher than **CoAP** transaction bytes due to 6LoWPAN and **CoAP** header compression.
- **CoAP** packet can be sent in single IEEE802.15.4 frame without fragmentation.
- Less bytes $\rightarrow$ lower power consumption and longer lifetime for **CoAP**.
Authors introduce an end-to-end IP based architecture that integrates CoAP over WSN with HTTP web application using a gateway.

System designed for greenhouse monitoring, but only a prototype implemented here!
- Contiki gateway attached to Linux machine via USB.
- As a prototype, application server and CoAP data collection functionality are in the same machine.
- Web client sends requests for WSN resources to Web server in gateway using HTTP.
- Web server retrieves resource data either from database (a gateway caching mechanism) or from the CoAP client.

- Web server either requests 'fresh' data from the WSN or receives data from the CoAP client (subscribe/publish) triggered by changes in resource at the CoAP server. {Web server bypasses database in both cases.}

- Authors use GWT (Google Web Toolkit) to develop Web application.
Since CoAP client receives WSN data in JSON, storing documents as JSON in Apache CouchDB provides RESTful API.

- Implementation was NOT tested under high frequency conditions.

- Authors worry about database caching mechanism becoming the bottleneck!
CoAP Client

- **libcoap** CoAP client communicates with the WSN.

- Since Contiki support for observations was not yet available, CoAP client does not handle publish packets from mote server.

- CoAP client adds timestamp to JSON data to support historical web server requests.
Gateway Implementation

- Gateway does not provide proxy functionality that converts HTTP requests to CoAP and vica versa.
- Web server invokes CoAP client with HTTP request parameters \(\rightarrow\) gateway is not transparent to the application and to the WSN.
- Gateway needs proxy functionality to support complicated operations such as observations.
Conclusions

- Authors provide IoT community with CoAP vs HTTP measurements that show power improvements from the μIP stack.

- Prototype gateway is a 'proof-of-concept' that matched the CoAP functionality built into Contiki in 2011.

- Paper encouraged proxy development.
Critique

- This is a good short paper ⇒ IPSN is a respectable conference in sensor area.
- CoAP explanation is clearer than in previous paper.
- There are several grammar/typo mistakes in the paper.
- Performance results could have included more than just power.