Debugging the Internet of Things: A 6LoWPAN/CoAP Testbed Infrastructure

Daniel Bimschas, Oliver Kleine, and Dennis Pfisterer

Institute of Telematics, University of Lübeck
Ratzeburger Allee 160, 23538 Lübeck, Germany
{bimschas,kleine,pfisterer}@itm.uni-luebeck.de

Abstract. This paper is based on two fundamental assumptions about a future Internet of Things (IoT): i) The amount of wireless, resource-constrained devices will outnumber the amount of devices in the current internet by several orders of magnitude and ii) those devices will be connected to the Internet over multi-hop wireless links. We argue that the experimental validation in testbeds is imperative to make those networks robust. However, there are only limited means to support researchers in "debugging" the actual communication on the wireless medium and often developers can only guess why their protocols don't work in a given environment. In this paper, we present such a framework which extends the WISEBED testbed federation. Our contribution allows an easy-to-use browser-based experimentation and evaluation of wireless multi-hop protocols in all WISEBED-compatible testbeds (nine testbeds with 1000 sensor nodes and the SmartSantander [17] smart city testbed which will offer up to $20,000$ IoT devices). Using a generic packet tracking framework for multiple platforms, researchers can easily detect hotspots and bottlenecks in the network and follow the routes of individual packets as they are forwarded. Experiment configurations can be shared on the web so that experiments can easily be repeated to verify published results. We demonstrate the usability of our approach by means of a real-world use-case.

Keywords: Experimentally-driven Research, Testbeds, Internet of Things, 6LoWPAN, CoAP.

1 Introduction

This paper is based on two fundamental assumptions about a future Internet of Things (IoT): i) The amount of wireless, resource-constrained devices will outnumber the amount of devices in the current internet by several orders of magnitude and ii) those devices will be connected to the Internet over

X.-Y. Li, S. Papavassiliou, S. Ruehrup (Eds.): ADHOC-NOW 2012, LNCS 7363, pp. 207–220, 2012.

multi-hop wireless links using standardized protocols such as 6LoWPAN[1] and the Constrained Application Protocol (CoAP)[2].

The integration of resources and services available in the traditional Internet with novel real-world services offered by IoT devices such as sensor nodes, gives rise to a completely new class of applications. However, the development of applications exploiting this combined infrastructure is cumbersome and difficult. This is due to the massively distributed nature of these networks combined with the heterogeneity of the devices, severe resource-constraints, and the difficulties of wireless multi-hop networking. In the past, simulations were a predominant means to test and optimize networking protocols. However, especially in wireless networking environments, many of these simulations lack realism and results strongly depend on the actual parameters. Apart from the fact that these were often not even reproducible [11, 20], the simulated environments and their implicit assumptions are also hardly comparable to any real-world setting.

As a result, IoT-testbeds such as MoteLab [19], Kansei [1], w-iLab.t Testbed [3], and WISEBED [4, 5] have become an important means to improve this situation by allowing researchers to evaluate the performance of protocols and applications [8] in real-world environments. This so-called experimentally-driven research has been instrumental in designing protocols that work efficiently in real-world settings. In the past virtually all experiments have been limited to the wireless network and there has been no or only very limited interaction with the Internet. If our two assumptions hold, a novel kind of testbeds is required that allows conducting experiments to exploit the merged infrastructure of the Internet and the Internet of Things. In this paper, we present

1. an extension to IoT testbeds to conduct experiments using standard Internet technologies such as IP and HTTP,
2. a framework for tracking packets in the wireless network, and
3. a tool for controlling, managing, and evaluating experiments using JavaScript only.

For our extension, we have chosen the WISEBED testbed federation, which, in contrast to virtually any other testbed, offers a well-defined web service API to allow for such extensions. Our contribution allows an easy-to-use browser-based experimentation and evaluation of wireless multi-hop protocols in all WISEBED-compatible testbeds (nine testbeds with 1000 sensor nodes and the SmartSantander [17] smart city testbed which will offer up to 20,000 IoT devices). Using a generic packet tracking framework for multiple platforms, researchers can easily detect hotspots and bottlenecks in the network and follow the routes of individual packets as they are forwarded. In addition to this, we present a reverse proxy that allows researchers to test the integration of WSNs into the Internet.

[1] 6LoWPAN [9] is a lightweight IPv6 adaptation layer allowing resource-constrained sensors to exchange IPv6 packets with the Internet.

[2] CoAP [15] is a draft by IETF's CoRE working group and provides a lightweight alternative to HTTP using a binary representation and a subset of HTTP's methods (GET, PUT, POST, and DELETE).

The remainder of this paper is structured as follows. Section 2 introduces our framework and discusses the extension to the WISEBED testbed federation, the reverse proxy architecture, and the packet tracking infrastructure. Section 3 presents a use-case in optimizing a 6LoWPAN- and CoAP-based application using our approach. Finally, Section 4 concludes this paper with a summary.

2 Architecture

Figure 1 depicts the extension to the WISEBED testbed federation (described in detail in Section 2.1), the reverse proxy architecture (Section 2.2) and their relationship with the overall testbed architecture. Section 2.3 discusses how packet tracking can be used on this architecture to help optimizing network protocols.

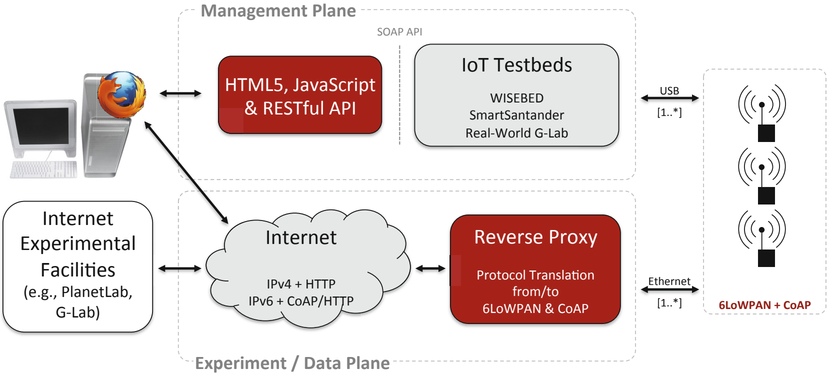


Fig. 1. Framework components overview

2.1 Testbed Extension

A variety of different testbeds for wireless sensor networks has been developed in the past that are now available to researchers. Each testbed has different characteristics and specific benefits [8]. However, the majority of them are also hardy extensible due to a proprietary user interface. These are often web-based and can only be operated by humans which prevents automated experimentation or extensions. An exception to this is the WISEBED [4, 5] testbed federation, which has defined a set of SOAP-based web service APIs that separate the interface of a testbed from the actual backend implementation and allows the implementation of generic clients and tools.

Consequently, we have chosen to build upon the WISEBED APIs to implement our extension. As a result, our extension is compatible with all WISEBED-compatible testbeds where currently several thousand sensor nodes from more than 10 different vendors are available. These APIs allow creating web-, console- and desktop-based clients with the ability to exchange messages with sensor nodes via their serial interface (e.g., to parameterize nodes to explore the parameter space in an experiment) as well as to reset crashed nodes, reprogram nodes, send control commands, etc. This allows a very high degree of interactivity with the experiment in which experimenters can control the experiment at runtime. For an in-depth overview, we refer the reader to the web page of the WISEBED project (http://wisebed.eu).

The goal of our extension is twofold: On the one hand, we provide a web-friendly interface to testbeds based on HTTP web services instead of SOAP-based ones to allow a broader acceptance. On the other hand, the goal was to build a modern web-based user interface that support browser-based experimentation (e.g., for packet tracking as discussed in Section 2.3). The motivation for this extension is the observation, that only a very limited subset of users was actually creating custom experimentation clients based on the SOAP API but instead used the simple ones shipped by the WISEBED consortium.

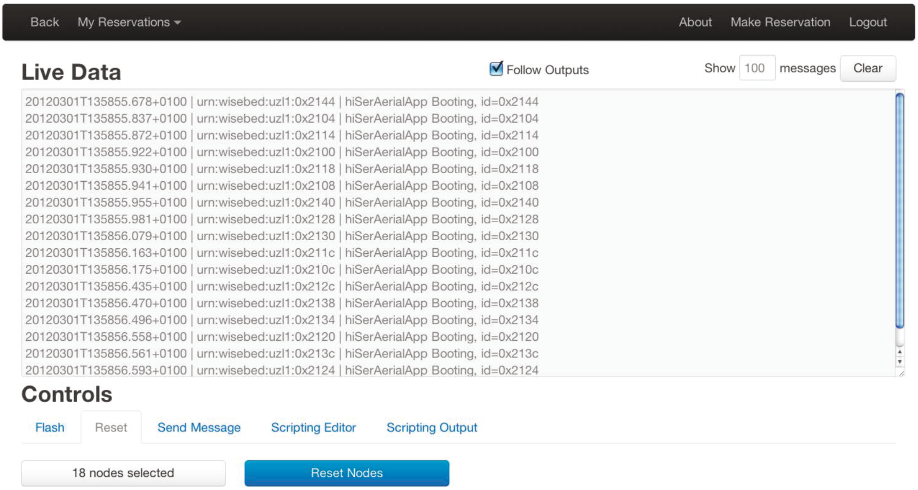
Additionally, the support for SOAP-based web services in languages such as JavaScript, Ruby, Python or others is limited and they often lack support for the WS-I web service interoperability standards. However, all these languages excel in their support for HTTP-based web services, also known as RESTful HTTP web services [7]. The extension presented here is based on easy-to-use technologies such as JavaScript, AJAX and JSON that allow users to create browser-based experiments with a very flat learning curve. In the remainder of this section, we now first introduce the RESTful HTTP-based web service API and then present the architecture of the new web-based user interface that uses this API and enables browser-based scriptable experimentation control. An example for such a scriptable execution is packet tracking to debug wireless communication as discussed in Section 2.3.

RESTful HTTP-Based Web Service API. The new HTTP-based web service API adheres to the REST architecture style and provides the same functionality as the SOAP-based APIs. Scripting experiments is now possible using virtually any interpreted or compiled programming language that supports HTTP and JSON (JavaScript Object Notation), thereby overcoming the issues with the SOAP-based APIs. Operations such as authentication, reservation, reprogramming, and resetting nodes are done by sending simple HTTP GET, PUT, POST and DELETE requests. Data exchange with sensor nodes is realized via *WebSockets* that allow bidirectional socket-like data exchange on top of, e.g., HTTP and avoid the problems of polling in AJAX-based web applications.

Both, the reference implementation and the documentation of the RESTful APIs are available as open-source projects at https://github.com/wisebed/rest-ws. The component is designed as a proxy mediating between the HTTP-based API and the SOAP-based APIs and is hence able to serve *all* WISEBED testbeds.

Web-Based User Interface Supporting Experiment Scripting. In addition to the HTTP-based API we developed a graphical user interface called

WiseGui that is solely based on cutting-edge HTML5 technologies. This includes new JavaScript and CSS features that are being subsumed under this standard which is already supported by all major browsers. The WiseGui already includes all required features for conducting experiments such as signing up, logging in, making reservations, ”connecting” to a reservation, programming and resetting nodes, sending messages to nodes, and displaying messages from nodes in a terminal-like window (cf. Figure 2). WiseGui is available for usage by everyone at http://wisebed.itm.uni-luebeck.de but may also be deployed on local testbed installations or embedded into other web sites.



☑ Follow Outputs

Show 100 messages

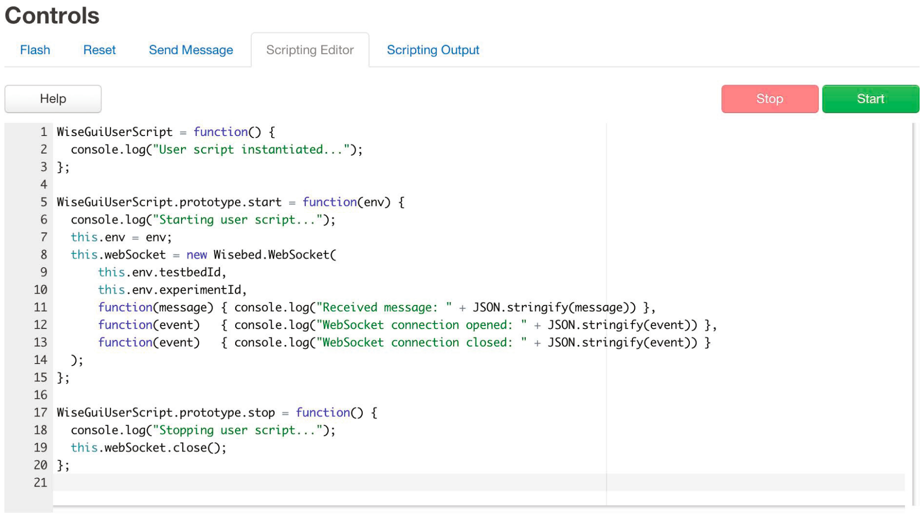
```
20120301T135855.678+0100 | urn:wisebed:uzl1:0x2144 | hiSerAerialApp Booting, id=0x2144
20120301T135855.837+0100 | urn:wisebed:uzl1:0x2104 | hiSerAerialApp Booting, id=0x2104
20120301T135855.872+0100 | urn:wisebed:uzl1:0x2114 | hiSerAerialApp Booting, id=0x2114
20120301T135855.922+0100 | urn:wisebed:uzl1:0x2100 | hiSerAerialApp Booting, id=0x2100
20120301T135855.930+0100 | urn:wisebed:uzl1:0x2118 | hiSerAerialApp Booting, id=0x2118
20120301T135855.941+0100 | urn:wisebed:uzl1:0x2108 | hiSerAerialApp Booting, id=0x2108
20120301T135855.955+0100 | urn:wisebed:uzl1:0x2140 | hiSerAerialApp Booting, id=0x2140
20120301T135855.981+0100 | urn:wisebed:uzl1:0x2128 | hiSerAerialApp Booting, id=0x2128
20120301T135856.079+0100 | urn:wisebed:uzl1:0x2130 | hiSerAerialApp Booting, id=0x2130
20120301T135856.163+0100 | urn:wisebed:uzl1:0x211c | hiSerAerialApp Booting, id=0x211c
20120301T135856.175+0100 | urn:wisebed:uzl1:0x210c | hiSerAerialApp Booting, id=0x210c
20120301T135856.435+0100 | urn:wisebed:uzl1:0x212c | hiSerAerialApp Booting, id=0x212c
20120301T135856.470+0100 | urn:wisebed:uzl1:0x2138 | hiSerAerialApp Booting, id=0x2138
20120301T135856.496+0100 | urn:wisebed:uzl1:0x2134 | hiSerAerialApp Booting, id=0x2134
20120301T135856.558+0100 | urn:wisebed:uzl1:0x2120 | hiSerAerialApp Booting, id=0x2120
20120301T135856.561+0100 | urn:wisebed:uzl1:0x213c | hiSerAerialApp Booting, id=0x213c
20120301T135856.593+0100 | urn:wisebed:uzl1:0x2124 | hiSerAerialApp Booting, id=0x2124
```

18 nodes selected

Fig. 2. WiseGui screenshot showing streamed output from the nodes’ serial interface

Apart from basic experimentation support, WiseGui has the outstanding feature to allow experimenters to script experiments directly in the browser by writing JavaScript in an embedded editor (cf. Figure 3). This completely eliminates the need for any client-side software installation. Scripts are executed right in the same browser window as WiseGui and can react to incoming messages from nodes, can generate messages to be sent to nodes, or can invoke any of the aforementioned functionalities to control the experiment (e.g., reprogram nodes, send configuration parameters, or reset nodes). Furthermore, scripts may modify the Document Object Model (DOM) of the web page. This for instance allows creating live visualizations of experiment data or updating tables displaying aggregated data. A use-case for this feature is introduced in Section 3.

A major issue with experimental facilities is repeatability of experiments by other researchers, e.g., to verify results published in papers. To facilitate such endeavors, the WiseGui enables “importing” of experiment configurations from any web page. Such a configuration contains a description of which nodes are



```
WiseGuiUserScript = function() {
  console.log("User script instantiated...");
};

WiseGuiUserScript.prototype.start = function(env) {
  console.log("Starting user script...");
  this.env = env;
  this.webSocket = new Wisebed.WebSocket(
      this.env.testbedId,
      this.env.experimentId,
      function(message) { console.log("Received message: " + JSON.stringify(message)) },
      function(event)   { console.log("WebSocket connection opened: " + JSON.stringify(event)) },
      function(event)   { console.log("WebSocket connection closed: " + JSON.stringify(event)) }
  );
};

WiseGuiUserScript.prototype.stop = function() {
  console.log("Stopping user script...");
  this.webSocket.close();
};
```

Fig. 3. WiseGui script editor

to be programmed with which binary image. Figure 4 shows an example defining two sets of nodes to be programmed with different binary images. The first set is determined by the URL query string `?capability=pir&filter=0x21` and consists of all nodes that have a passive-infrared sensor (PIR) and contain the string `0x21` in their description. The second set is defined likewise for temperature sensors. Upon passing the URL of this configuration to the WiseGui it will first resolve the sets of nodes, then download the binary images from the URL defined by the (relative) path in `binaryProgramUrl` and finally flash the individual node sets with the correct image file.

A future addition will be to include a URL to an evaluation script. This allows researchers to include links in publications pointing to the configurations that were used to generate the published results. As such a configuration includes the binary program images, the mapping of programs to nodes, and the evaluation script other researchers can repeat each experiment by just pasting this link into the WiseGui.

2.2 Reverse Proxy

As mentioned in the introduction, resource-constrained IoT devices are typically not capable of providing direct IP connectivity or even web services via HTTP. Because of this, an IPv6 adaption layer (6LoWPAN [9]) and a lightweight alternative to HTTP (CoAP [15]) have been developed. For integration into the Internet, a (transparent) protocol conversion is required. In the context of the EU-project SPITFIRE, we have developed such a reverse proxy architecture [12], which has been extended for the work presented here. This so-called Smart Service Proxy (SSP) acts as a border device between the Internet and an IoT

```
{ "configurations" : [
    {
      "nodeUrnsJsonFileUrl" : "http://wisebed.itm.uni-
          luebeck.de/rest/2.3/uzl/experiments/nodes?
          capability=pir&filter=0x21",
      "binaryProgramUrl"    : "bin/JN5148/Nodes.bin"
    }, {
      "nodeUrnsJsonFileUrl" : "http://wisebed.itm.uni-
          luebeck.de/rest/2.3/uzl/experiments/nodes?
          capability=temperature&filter=0x21",
      "binaryProgramUrl"    : "bin/JN5148/Gateway.bin"
    }
  ]
}
```

Fig. 4. Experiment configuration

network (e.g., a sensor network) as illustrated in Figure 1. The SSP provides a transparent protocol translation on different layers: on the network layer, it converts IPv6 to 6LoWPAN and on the transport and application layer, it converts all three protocol combinations {HTTP, TCP, IPv4}, {HTTP ,TCP, IPv6}, and {CoAP, UDP, IPv6} to {CoAP, UDP, 6LoWPAN}.

The core of the SSP is based on Netty [10], a Java framework for asynchronous network I/O. Internally the SSP is organized into modules with different responsibilities. These include protocol conversions, mime-type conversions, caching, compression, and more. These modules are arranged as a stack and protocol packets are passed from bottom to top through the modules for analysis, processing and response creation and back from top to bottom afterwards.

The topmost module is called backend because it either provides the requested data itself or obtains the data from a network behind. In our case it is responsible for application layer protocol translation from HTTP to CoAP and vice versa and forwarding the translated request to the sensor node, resp. the translated response to the Internet client.

The network-layer conversion allows a direct integration of IoT devices into the Internet and enables a seamless exchange of IP packets. The application-layer conversion provides integration on a service-level, i.e., integration into the World Wide Web. The service-level integration offered by the SSP is agnostic of the actual version of IP and accepts HTTP requests over IPv4 and IPv6. This is of special importance as IPv4 is still the predominant protocol on the Internet.

Regarding the data link layer, the SSP provides Ethernet on the Internet side and a 802.15.4 radio on the WSN side. Since data link layer protocols may vary on the route between client and server this is not exactly a protocol conversion but is mentioned for the sake of completeness. Note, that the IPv6/6LoWPAN conversion is not part of the publicly available source code at https://github.com/ict-spitfire/smart-service-proxy. Actually, this is realized using an external box having two data link layer interfaces. Its ethernet

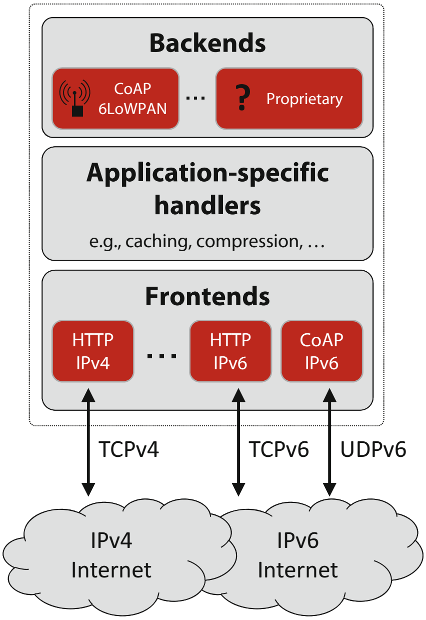


Fig. 5. SSP's protocol conversion architecture

interface is connected to the SSP whereas its 802.15.4 radio interface is the default gateway for the WSN built up of sensor nodes. The network connection between both devices is realized using a dedicated IPv6 net. From routing perspective the SSP acts as default gateway for the external box whereas the external box acts as default gateway for the WSN. By this means, we have three nets involved. The Internet (connected to the SSPs eth0), the net 2001:638:70a:c002::/64) between the SSP (eth1) and the external box (eth) and 2001:638:70a:c005::/64 for the WSN. However, for the sake of understandability the combination of both devices is refered to as SSP in the following.

Network- and Transport-level Conversion. Converting IPv6 to 6LoWPAN is a straightforward thing to do and thus not further introduced here. However, on transport level, things are more complex since sensor nodes are too resource-constrained to support TCP [14]. Consequently, the SSP intercepts TCP connection attempts and accepts them on behalf of the sensor node. From a client's perspective the TCP connection appears to be end-to-end to the sensor node.

Assume a sensor node with the IPv6 address 2001:638:70a:c005::2 running a CoAP web server on top of UDP. The gateway to the Internet is 2001:638:70a:c005::1 (the IPv6 address of the SSPs 802.15.4 interface). Furthermore let's assume an Internet client using a web browser with HTTP over TCP.

The browser tries to establish an end-to-end TCP connection with the sensor node. This connection attempt is intercepted by the SSP as illustrated in Figure 6. Each arrow represents the transfer of a TCP packet, either between

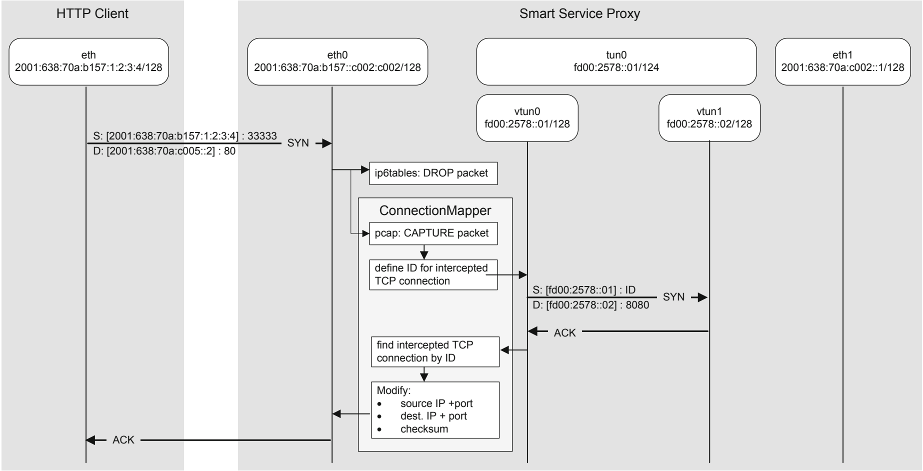


Fig. 6. Intercepting a TCP connection attempt

two sockets or with method calls within the application. The source port 33333 is just an example but in fact randomly chosen by the HTTP client. The TCP connection attempt starts with a SYN message from the client.

The original packet is dropped using the following ip6tables Linux firewall rule: `ip6tables -A FORWARD -i eth0 -j DROP`. PCAP (see [18] for Linux and [13] for Windows OS) captures the packet before being dropped and passes it to the so-called *Connection Mapper*, which is responsible for the TCP interception handling. The *Connection Mapper* is a software component of the SSP that uses jNetPcap [16] to access the operating system-specific libpcap library [18]. Each TCP connection is uniquely identifiable by the quadruple {source IP, source port, destination IP, destination port}. The Connection Mapper memorizes the quadrupel of the captured TCP connection and assigns an ID (range 1 to 65535) to the quadrupel.

In order to establish a virtual TCP connection without the need for re-implementing TCP, virtual interfaces are used. For this, the SSP creates a virtual IP interface (tunX) assigned with a /124 IPv6 network from the unique local address space. After assigning the original TCP connection attempt (SYN from Internet client) with an ID, the Connection Mapper uses this ID as source port to establish a local TCP connection between two virtual tun interfaces (vtun0 and vtun1 in Figure 6). Since the port equals the ID, the Connection Mapper can always map the ID to the quadruple.

Service-level Conversion. A client in the Internet has several options to contact a CoAP-based RESTful web service [7] offered by a sensor node using either {HTTP, TCP, IPv4}, {HTTP ,TCP, IPv6}, or {CoAP, UDP, IPv6}. The first and the second require a conversion from HTTP to CoAP. In addition, TCP must be "split up" to UDP (port addressing) and CoAP (reliability) and for

IPv4 also a conversion to IPv6 is required. In the following, we explain these conversions in detail.

To allow an IPv4-based access to CoAP/6LoWPAN-based services, a mapping between IPv4-based HTTP URLs (e.g., http://127.0.0.1/some/service) and IPv6-based CoAP/6LoWPAN-based URLs (e.g., coap://[2001:638::CDEF]/some/service) is required. Our approach is based on a wildcard DNS entry that maps a sub-domain to a single IPv4 address. We resolve *.coap2.wisebed.itm.uni-luebeck.de to the IPv4 address of the SSP. The IPv6 address of a sensor is encoded in the hostname of the sub-domain by replacing each colon with a minus.

A client would then set the *Host:* field of the HTTP request to http://2001-638–CDEF.coap2.wisebed.itm.uni-luebeck.de and the SSP uses this field to extract the IPv6 address of the sensor node. Converting HTTP to CoAP is done as follows: The SSP translates the HTTP requests to CoAP requests, forwards them to the corresponding sensor nodes, waits for the CoAP response, translates it into a HTTP response, and sends the response to the client.

As described in [15] the protocol translation regarding the mapping between HTTP header fields and CoAP options is quite straightforward and thus not further introduced here. The main pitfall is the underlying transport protocol which is the connection based TCP for HTTP and the connectionless UDP for CoAP. CoAP provides optional reliability and by this means provides a core functionality of TCP based on UDP. Thus, the SSP must keep the intercepted TCP connection and disperse its functionality on UDP and CoAP. CoAPs reliability bases on acknowledgements (ACKs) to be sent in answer to a received message. The ACK message can either contain a response code and payload (a representation of the requested resource) or be empty, indicating that the server is willing to answer the request but needs more time to do so. However, if there was no non-empty CoAP response within a predefined period of time, the SSP sends a HTTP time-out message to the Internet client and a RST message to the CoAP server to cut the request processing of.

2.3 Packet Tracking

The development of applications for wireless sensor networks is cumbersome and error-prone, in particular due to the faulty wireless communication channels they typically use. This is especially painful when experimenting with new or evolving routing schemes in multi-hop environments. Often experimenters observe that packets sent over multiple hops get lost but the exact reason is unknown. Experienced researchers may guess the correct reason (e.g., the wireless channel is occupied due to excessive flooding or the like) and may even be capable to solve some of the causes. However, virtually always, developers must resort to ”println()-debugging” to be able to track down the root cause of the problem by manually analyzing debugging data received via the serial interfaces.

This situation is aggravated by the fact that some problems only occur in large-scale networks [6], which basically eliminates debugging on the developer's desk. In such situations, experimental facilities are helpful that allow large-scale experimentation. However, while supporting large-scale experiments, there is virtually no support beyond println()-debugging.

We propose a generic mechanism for tracking packets in a testbed to analyze where hotspots are, where the medium is heavily loaded or to track individual packets as they are forwarded in the network. Our approach is based on a contract between the wireless sensor nodes and an evaluation script. The nodes forward any activity on the wireless medium to the serial interface (i.e., received and transmitted packets). An evaluation script running in the WiseGui (cf. Section 2.1) will receive all data from all nodes in a testbed and can then run custom evaluations.

We have already implemented support for this kind of debugging in the iSense operating system and will be finalizing an implementation for the Wiselib [2] template library soon, which adds support for TinyOS, Contiki, and Scatterweb2 to name only a few. The packet format that nodes use on their serial port for this purpose is depicted in Figure 7. `TYPE` indicates the type of the packet (e.g. an IPv6 packet), `DIRECTION` indicates if a packet was received or sent, `SRC_MAC` and `DST_MAC` contains the 64-bit MAC addresses of sender and recipient of the packet and `PACKET` contains the packet itself. The node that emitted the trace packet can be determined through the metadata that is delivered together with the trace packet. An example of how this format can easily be used to determine communication hotspots is presented in Section 3.

```
+------+-----------+---------+---------+--------+
| TYPE | DIRECTION | SRC_MAC | DST_MAC | PACKET |
+------+-----------+---------+---------+--------+
```

Fig. 7. Packet Tracking format for the serial interface

3 Use-Case: Optimizing a CoAP Implementation

In preparation of this paper we found ourselves in exactly the same situation as described above – having an implementation of a standardized routing algorithm that wouldn't perform on the testbed, resulting in only a few nodes being reachable from the Smart Service Proxy. To verify our assumption that the poor routing performance resulted from very high traffic load in certain network regions we developed a simple JavaScript-based visualization application running in the WiseGui to find the communication "hot spots".

Therefore, we employed the packet tracking concept described earlier to generate trace packets for all packets being sent and received over the sensor nodes radio interfaces. The trace packets are forwarded to the visualization application running on the experimenters browser. The testbeds self-description is used to determine the physical positions of the individual sensor nodes and to draw them to a canvas accordingly. Upon every trace packet received a packet count for the individual node is incremented and the visualization is updated. Figure 8 shows a screenshot of the visualization. Nodes are displayed as circles containing the number of packets received and sent so far and the radius of the circle corresponds to the packet count.

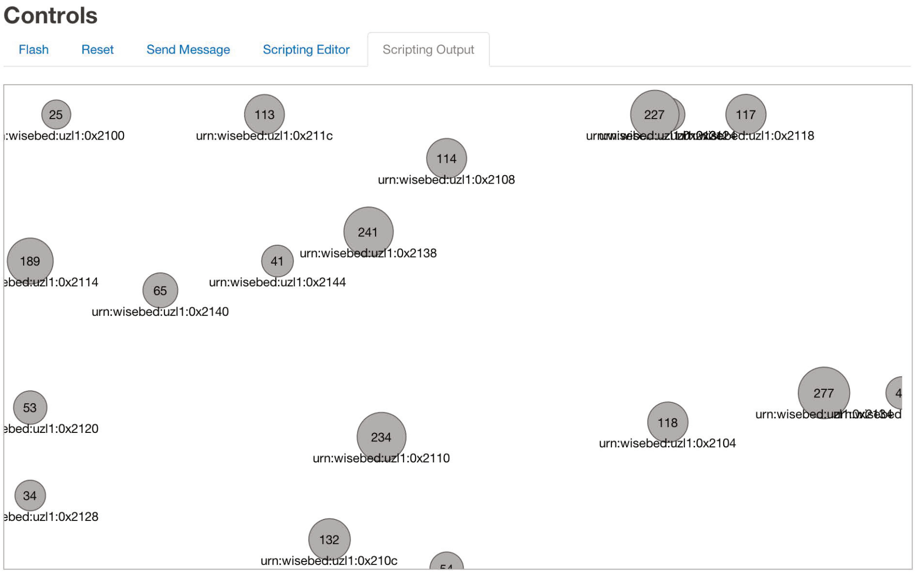


Fig. 8. Packet tracking visualization in the WiseGui

The experiment was executed on the WISEBED testbed at the University of Lübeck. The experiment configuration file, the binary images for the sensor nodes as well as the visualization script are available on http://goo.gl/0AZXl.

4 Conclusion

In this paper, we have motivated the need for appropriate testbed environments that support researchers in conducting experimentally-driven research. By appropriate, we mean that it is possible to debug the wireless communication efficiently. In the past, this was mainly done manually by sending data to the serial interface and by interpreting these results. However, very often this resulted in a lot of guessing about what exactly the source of the problem is. We argue that a testbed infrastructure should inherently support such functionality as some issues only occur at large-scale and cannot be tracked down on a desktop-scale deployment.

As a result, we have presented a framework for debugging the wireless communication in multi-hop networks by extending the WISEBED testbed infrastructure. The overall goal was to create an easy-to-use platform that also allows that other researchers can repeat experiments to verify published results. Our approach has been to use up-to-date web standards in order to be able to run and evaluate experiments in a browser without the need to install additional software. To achieve this, we have designed a RESTful HTTP-based web service API acting as a proxy for SOAP-based WISEBED testbeds and a web-based user interface called WiseGui that supports to script experiments. In addition, we have presented a generic packet tracking framework where nodes emit information about sent and received packets over their serial interface. This data is evaluated by such scripts and we have shown in our use-case how this technology can be used to optimize multi-hop protocols by pinpointing communication hotspots.

We strongly believe that the presented framework has the potential to fundamentally change the way we conduct experimentally-driven research and how results can be verified.

Acknowledgements. This work has been partially supported by the European Union under contract number ICT-2009-258885 (SPITFIRE) and by the Federal Ministry of Education and Research of Germany (Förderkennzeichen 01BK0905, Real-World G-Lab).

References

1. Arora, A., Ertin, E., Ramnath, R., Nesterenko, M., Leal, W.: Kansei: A high-fidelity sensing testbed. IEEE Internet Computing 10, 35–47 (2006)
2. Baumgartner, T., Chatzigiannakis, I., Fekete, S., Koninis, C., Kröller, A., Pyrgelis, A.: Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks. In: Silva, J.S., Krishnamachari, B., Boavida, F. (eds.) EWSN 2010. LNCS, vol. 5970, pp. 162–177. Springer, Heidelberg (2010), http://ewsn2010.uc.pt/, ISBN 978-3-642-11916-3
3. Bouckaert, S., Vandenberghe, W., Jooris, B., Moerman, I., Demeester, P.: The w-iLab.t Testbed. In: Magedanz, T., Gavras, A., Thanh, N.H., Chase, J.S. (eds.) TridentCom 2010. LNICST, vol. 46, pp. 145–154. Springer, Heidelberg (2011)
4. Chatzigiannakis, I., Fischer, S., Koninis, C., Mylonas, G., Pfisterer, D.: WISEBED: An Open Large-Scale Wireless Sensor Network Testbed. In: Komninos, N. (ed.) SENSAPPEAL 2009. LNICST, vol. 29, pp. 68–87. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-11870-8_6
5. Coulson, G., Porter, B., Chatzigiannakis, I., Koninis, C., Fischer, S., Pfisterer, D., Bimschas, D., Braun, T., Hurni, P., Anwander, M., Wagenknecht, G., Fekete, S.P., Kröller, A., Baumgartner, T.: Flexible experimentation in wireless sensor networks. Communications of the ACM 55(1), 82–90 (2012), http://doi.acm.org/10.1145/2063176.2063198
6. Exscal Research Group, Ohio State University: Extreme scale wireless sensor networking (2010), http://ceti.cse.ohio-state.edu/exscal/
7. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
8. Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., Razafindralambo, T.: A survey on facilities for experimental internet of things research. IEEE Communications Magazine 49, 58–67 (2011), http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6069710
9. Kushalnagar, N., Montenegro, G., Schumacher, C.: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational) (August 2007), http://www.ietf.org/rfc/rfc4919.txt

10. Netty Project: Netty is an asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers and clients, http://netty.io
11. Pawlikowski, K., Jeong, H.D.J., Lee, J.S.R.: On credibility of simulation studies of telecommunication networks. IEEE Communications Magazine 40(1), 132–139 (2002), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=978060
12. Pfisterer, D., Römer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., Hasemann, H., Kröller, A., Pagel, M., Hauswirth, M., Karnstedt, M., Leggieri, M., Passant, A., Richardson, R.: SPITFIRE: Toward a semantic web of things. IEEE Communications Magazine 49, 40–48 (2011), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6069710
13. Riverbed Technology: winpcap (2012), http://www.winpcap.org
14. Rothenpieler, P.: Poster abstract: Distributed protocol stacks for wireless sensor networks. In: 9th European Conference on Wireless Sensor Networks (EWSN 2012), Trento, Italy (February 2012)
15. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: Constrained application protocol (CoAP) (CoRE working group) (2011) Online version at, http://www.ietf.org/id/draft-ietf-core-coap-08.txt (November 01, 2011)
16. Sly Technologies: jNetPcap, http://jnetpcap.com
17. SmartSantander consortium: SmartSantander EU FP7 project (2010), http://www.smartsantander.eu/
18. TCPDUMP: libpcap, http://www.tcpdump.org
19. Werner-Allen, G., Swieskowski, P., Welsh, M.: Motelab: a wireless sensor network testbed. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN 2005. IEEE Press, Piscataway (2005), http://portal.acm.org/citation.cfm?id=1147685.1147769
20. Wittenburg, G., Schiller, J.: A quantitative evaluation of the simulation accuracy of wireless sensor networks. In: Proceedings des 6. Fachgespraechs "Drahtlose Sensornetze" der GI/ITG-Fachgruppe "Kommunikation und Verteilte Systeme", Aachen, Germany, pp. 23–26 (July 2007), http://page.mi.fu-berlin.de/wittenbu/research/wittenburg07quantitative.pdf