Masters Project Report

Enhancing a Wireless Control Testbed

Guillaume VALENTIS
Washington University in St. Louis
Engineering, Robotics Major
Paris, France
guillaume.valentis
@wustl.edu

ABSTRACT

This semester, my work on this project is to enhance the Wireless Sensor Network (WSN) testbed located in Jolley Hall of Washington University. The testbed is working with Time-Slotted Channel Hopping (TSCH) [1, 2] that can support a considerable number of nodes. The first part of my work consisted in a literature review about wireless security that can affect TSCH and WirelessHART systems. Many kinds of attacks can affect such a system: from hacking to physical tempering and stealing information. My main focus was the upgrade of the Operating System (OS) used on the testbed: currently the testbed runs under TinyOS which was last updated in 2012 with version 2.1.2 [3] and version 2.2 is still pending [4] and its last update was in 2014. The new OS decided to be used is ContikiOS, an open source embedded OS designed for low powered WSN. The project I was assigned is to setup a TSCH network on the testbed using ContikiOS.

Keywords

wsn; tsch; contikios

1. INTRODUCTION

Developments in WSN are revolutionizing our world. In the past, everything had to be wired and that requirement was inducing an increase in cost for materials (wires) and installation: more space to fit the wires and special installation and maintenance system. However, during the last decade, WSN started to develop and expand rapidly. New devices were created and new microcontrollers were designed with low power consumption and small sizes. With that, new OS and ways to communicate wirelessly appeared. IPv6 got a huge boost with the idea of giving each device its own IP address on the Web. Thus, the IPv6 Low power Wireless Personal Area Networks (6LoWPAN) was developed and later 6TiSCH appeared in order to link IEEE802.15.4e TSCH capabilities with 6LoWPAN.

DOI: ˜

ContikiOS [5, 6] is an open source OS designed for the internet of things and for low cost and low powered microcontrollers. This OS supports a wide range of microcontrollers and fully supports the new standards of IPv6 and IPv4, 6LoWPAN, Routing Protocol for Low-Power and Lossy Networks (RPL), TSCH. The OS is also frequently updated and maintained. ContikiOS is written in standard C and all the applications and core functions are also in C. ContikiOS also comes with an emulator (Cooja) to simulate sensor networks before running it on the hardware.

The testbed in Jolley Hall includes 70 TelosB connected to 43 RaspberryPi distributed over 3 floors (3^{rd}, 4^{th} and 5^{th}). Each of the RaspberryPi are connected to a switch, under the same subnet, which is connected to the WSN server. In order to work with the testbed, we connect on the server through ssh and from it we can directly burn the TelosB designated by their id.

This report is divided in five parts besides the introduction: Section 2 presents the goals of the project, Section 3 is about my research on the main security issues of a WSN, then Section 4 is about my work on ContikiOS and the testbed, Section 5 describes the difficulties encountered during my work and how they were solved. Finally, the conclusion and the future works are presented in Section 6.

2. GOALS

My goals for this project were to (1) list important security issues related to WSN and WirelessHART systems; (2) start working with ContikiOS on the testbed and see what changes would need to be made in order to make the new OS work; (3) implement a multi-hop communication systems using TSCH and ContikiOS on the testbed; (4) lastly my work is also to provide enough documentation in order to make this transition, from TinyOS to ContikiOS, as smooth as possible.

3. SECURITY

3.1 Main Issues

WSN are usually more sensitive to attacks than wired systems. It is easier to have a handheld device and to eavesdrop on the sensor network than to attach a physical device on a

communication cord.

There are six main threats that I was able to identify during the time I was working on security [7]:

- Wormhole
- De-synchronization
- Jamming
- Traffic analysis
- Spoofing
- Exhaustion attack

3.2 Wormhole

Wormhole attacks can threaten the stability of the whole system. If a node is inserted or a node is replaced with a wormhole that drops the packets, valuable information can be lost: no data update from a sensor to an actuator means no control over the system.

3.3 De-synchronization

De-synchronization of a node leads to communication disruptions from this node and all other routes passing from it. A re-synchronization needs some valuable time: either in energy resources or in data transition that is lost.

3.4 Jamming

Jamming is one of the most difficult and unpredictable threats for wireless networks. It leads to communication disruptions and also to de-synchronization. Many things can jam unintentionally a signal such as Wi-Fi, phones, Bluetooth, magnetic fields, but also intentionally: malicious people aiming to disrupt the system.

3.5 Traffic Analysis

In the legacy WirelessHART system NPDU header and DLPDU are un-encrypted: it is possible to listen to the traffic and learn about the routes used to transmit the data. traffic analysis can also lead to finding the new devices added to the network, the routines and the work peak hours.

3.6 Spoofing

The threat is the possibility to be able to join the network using the well known key and advertise.

3.7 Exhaustion

Such as Spoofing getting the well known key and using it to send fake packets to the neighboring nodes leads to an unwanted use of resources. If this kind of attack is used with many devices it can also be transformed to a DOS attack.

4. CONTIKIOS

4.1 Working Environment

To start developing for ContikiOS two choices are presented: downloading a ready to use virtual machine (Instant Contiki) based on Ubuntu 14.04 or installing some packages and cloning the git repository of ContikiOS.
For some platforms it is easier to choose the first option as many tools might be difficult to get installed by someone unfamiliar with it. In our case we are using TelosB motes and we will just need a toolchain (gcc-msp430) to compile the code and an application to burn the hardware: tos-bsl (a TinyOS tool to burn the binary file on the hardware) as I want to minimize the changes of tools of the current system if possible.

4.2 First example

First step in this work was to see if the tools installed could work properly on both the TelosB mote and the testbed. I used the basic hello-world example given in ContikiOS. The compilation of the example outputs a .sky file that can be converted to a .ihex file which is the same format as the TinyOS binary files. The example worked on both a single TelosB mote connected to my computer and the testbed. This test allowed me to be clear on the fact that, except for the OS there wouldn't be any significant change on the way the other tools are used to work on the testbed.

4.3 Wireless communication and TSCH

On ContikiOS my work continued with testing TSCH communication between some nodes. I used the Rime communication stack and TSCH:

- Coordinator node #1
- Node #2 sending to node #1
- Node #3 sending to node #1
- Node #4 sending to node #2

While the coordinator was able to receive messages from node #2 and #3, node 2 wasn't getting any messages from node # 4.

After this result, I didn't go any further. Indeed, after exchanging emails with Simon Duquennoy, who is working on ContikiOS and TSCH at the Swedish Institute of Computer Science (SICS), he told me that Rime is moving to deprecation and is almost not used anymore on ContikiOS.

4.4 6TiSCH - RPL

ContikiOS offers a wide range of tools and a complete net stack given that we understand the configuration and enable/disable what we need and don't need.
I continued my work around TSCH, as my main goal was to make it work properly, using 6TiSCH and RPL. I created a sample code of a receiver and multiple sender nodes. I fixed their IPv6 according to their node ID in order to differentiate them more easily.

Here I am using the terms of receivers and transmitters, but it just means that the receiver is not sending any messages, whereas all the transmitters are able to receive messages if they are recipients.

Before enabling TSCH I started experimenting just with 6LoWPAN and RPL to understand the configurations. Once I was clear that multi-hop and data transmission was working, I enabled TSCH.

For the first experimentation I used 2 nodes: one receiver and one transmitter.
Then I added three more transmitters to test the multi-hop

from the transmitter to the recipient.

All those experiments were first tested using the emulator Cooja, where I placed different nodes with different spatial configurations to test that my code was working. Then, in the case of one receiver and one emitter, I did it with the TelosB motes linked to my computer. The last test, with multiple senders, was done on the testbed after confirming with Cooja that it was working as expected.

Lastly both the 6LoWPAN - RPL and 6TiSCH - RPL experiments were run on the testbed in a similar configuration as the simulation. I was able to observe some dissimilarities between the emulator and the testbed: First, some of the packets sent from the more distant nodes were not able to arrive at destination. Then, sometimes, the nodes weren't able to create a stable schedule for the TSCH to work, some packets weren't able to arrive at destination.
Indeed, the Cooja simulator was emulating an optimal environment for wireless communication; no interferences external sources.

5. DIFFICULTIES

The first time I tested TSCH, I worked with Rime because RPL and 6TiSCH were too heavy to use for the ROM of the TelosB: the ROM memory was overflown by 10KB.

But, as stated in Section 2, I had to implement a multi-hop communication system using TSCH.
Simon Duquennoy tipped me to disable everything that I didn't need. Some features of ContikiOS are enabled by default even if you don't use them.
Features disabled at first and that can be easily found in the configuration file:

- TCP
- UDP checksum
- Debug
- Logs (both from ContikiOS and TSCH)

Even with all those heavy features disabled, I was still short of about 3.8KB of ROM.

I realized that ContikiOS was abusing the *printf* function from the stdio library. One must know that this function is about 4KB in ROM memory for microcontollers. On a system like the TelosB that has 48KB of ROM, *printf* uses about 8.3% of it. The 6TiSCH stack and RPL on ContikiOS are already heavy and we also have to consider the core part of the OS and our application.

To do that, I simply defined an empty *printf* and *sprintf* in the header configuration file:

- #define printf(...)
- #define sprintf(...)

At that point, without this oversized function, I was able to compile and run the application. I am still concerned about the fact that the ROM memory might be used at 99% or more, which leaves small to no more memory available for a more complex application.

In order to be able to see what was happening I replaced the *printf* with a simple *lprint* function that just outputs the string given through the USB port. Adding a simple and light integer to string function, I was able to send back valuable information, such as the message id and the sender's id. And for all the other string operations I used the basic string manipulation functions provided by the std libraries (*strcpy, strcat*).

When I was able to compile and test the code on Cooja, I noticed that the nodes couldn't communicate with each other. After some work, I found out that no schedule was created before starting to send the messages. To solve the issue I added a delay of 1 minute before the first message transmission. After that, all the nodes were able to transmit their messages to the receiver.

6. CONCLUSION

This is a very interesting Masters Project and my work on improving the testbed was very tricky with lot of various problems to solve. First it was about multi-hop issues. Then, I had to face a ROM overflow problem before finally being able to make the system work as it was expected.
During this semester, the work with wireless systems opened to me new horizons. We are using more and more WSN and we need a safe and reliable way to make them communicate together without interfering with one another.

There is still a lot of work to do on what I started.
As I stated in Section 5, the memory of a TelosB is used at more than 99%, which means that there is almost no memory left for future applications. It should be a necessity to find other features to disable (like TCP) and also functions to remove completely or replace, with a lighter version, from the system (such as *printf*).

Lastly, during my experimentations I was using the minimal schedule that was provided with the TSCH stack, and I didn't know how many nodes this schedule could support, or how long it would take to obtain a stable schedule.

7. APPENDIX

The tutorials I made can be found on the CPSL Wiki page: Tutorials.
You can download my work on the same page under the Download section: Download.
Direct links:

- 6TiSCH application with one receiver and four transmitters: 6TiSCH.tar.gz
- ContikiOS app used as a replacement for *printf*: lprintsky.tar.gz

8. REFERENCES

[1] S. Duquennoy, B. Al Nahas, O. Landsiedel and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled tsch," *Proceedings of the International Conference on Embedded Networked Sensor Systems*, Nov 2015.

[2] T. Watteyne, M. Palattella and L. Grieco, "Using ieee 802.15.4e time-slotted channel hopping (tsch) in the internet of things (iot): Problem statement," *IETF, https://tools.ietf.org/html/rfc7554*, May 2015.

[3] TinyOS, "TinyOS Release 2.1.2," 2014. http://tinyos.stanford.edu/tinyos-wiki/index.php/Installing_TinyOS.

[4] TinyOS, "TinyOS Release 2.2," 2014. https://github.com/tinyos/tinyos-main/milestone/1.

[5] A. Dunkels and B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004.

[6] ContikiOS, "ContikiOS web page," 2017. http://www.contiki-os.org/.

[7] Shahid Raza, "Secure Communication in WirelessHART and its Integration with Legacy HART," 2010. http://soda.swedishict.se/3799/1/T2010_01.pdf.