Demo: A Complete Framework for Programming Event-Driven, Self-Reconfigurable Low Power Wireless Networks

Marcin Szczodrak
Columbia University
msz@cs.columbia.edu

Luca Carloni
Columbia University
luca@cs.columbia.edu

Abstract

We present a complete framework to design and deploy adaptive low power wireless networks. The framework consists of Fennec Fox, a four-layer network protocol stack, and Swift Fox, a high-level programming language. At run-time, Fennec Fox dynamically reconfigures services running on the network protocol stack layers using a library of modules optimizing a layer's performance with respect to some metric (delay, power consumption, *etc.*). While network reconfiguration is triggered by sensing or timer events, policies specifying how a network should be reconfigured when given events occur are programmed in Swift Fox at design time. We discuss a network that reconfigures its communication services to support 3 scenarios and that was tested on mica2, intelMote2, and telosB architectures, the last one requiring 21KB of ROM and 2KB of RAM.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

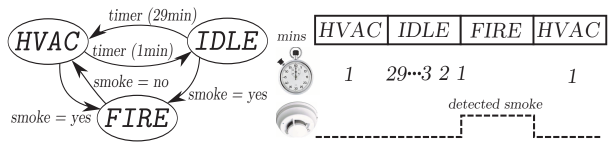
Design, Experimentation, Measurement

Keywords

Adaptive Low Power Wireless Networks, Sensor Networks, Programming Languages, Network Protocol Stack

1 System Description

Over the last decade we have observed an increasing number of applications using low power wireless networks (LPWN) integrated with sensors and actuators. As applications become more heterogeneous, a network with a single communication service supporting all types of applications becomes inefficient, and quickly consumes the limited network resources. To address this problem, various applications, network routing, and medium access control (MAC) protocols, as well as designs of low power radio RF transceivers have been prototyped and published. Researchers have shown customized solutions (*e.g.,* a protocol) through which overall system performance with respect to some metric (*e.g.,* power consumption [1], reliability [2], latency [3], *etc.*) becomes superior than with other general-purpose, standard approaches. With this in mind we believe that to achieve a better overall network performance and resource utilization, LPWN should dynamically customize its service by executing modules that can meet application requirements and/or overall system goals. To build such network we need mechanisms switching execution of the modules at run-time, and tools supporting programing the network reconfiguration logic at design time.



(a) System FSM Model (b) System Trace

Figure 1. Multi-State Self-Reconfigurable Network

Example of a Multi-task Network Problem. In many scenarios, *e.g.,* smart buildings and health-care, LPWN provides communication infrastructure to various applications. For example, in smart building, the same network infrastructure might be used for control of an HVAC system, occupancy monitoring, safety, security, and patient monitoring. Each of these applications has its own expectations with respect to the communication services: reliability, latency, power consumption, just to name a few.

To demonstrate the problem we describe a simplistic LPWN system used for HVAC and safety applications. Every half-hour the HVAC application collects for a period of one minute data with building's temperature and occupancy, estimated with the temperature and camera sensors. To ensure long network's lifetime, the application requires a power-efficient data-collection network. In case of emergency, for example a fire, the application takes pictures of the unsafe area and streams them to an emergency response team. To ensure quick reconnaissance, the application requires low-latency, high-throughput, point-to-point channel.

We translate the presented LPWN system into a finite state machine model. In this model we have 3 states: *HVAC*, *fire*, and *idle*, as shown on Figure 1(a). Every 30 minutes the network transitions between the *HVAC* and the *idle* states. To save overall system energy consumption, the network spends most of its time in the *idle* state where it operates in a low-power listening mode. When smoke sensors detect a fire the network reconfigures into the *fire* state. Once the fire is over, the network transitions to the *HVAC* state. Figure 1(b) shows an example of a network reconfiguration trace, where timer based events are used to transition between the *HVAC* and *idle* states, and smoke sensor events are used to transition between the *fire* state and two others. We introduce a new programming language to set up a self-reconfigurable LPWN.

Programming Language. Swift Fox is a declarative programming language to model LPWN network states and policies controlling event-based state transitions. A programmer defines all network states, where each state consists of one or more network-stack configurations. Each configuration defines modules that together execute some task, *i.e.,*

```
1  # Definition of system states: conf <state_id> {<application> <network>
2  #                    <network_addr> <qoi> <mac> <mac_addr> <radio>}
3  conf hvac {temp(3) ctpf() fixed(16) none sMac(10) fixed(16) cc2420(1)}
4  conf idle {none none none none none none none}
5  conf fire {pic(2, 1) p2p(1) fixed(8) none setRate() fixed(8) cc2420(1)}
6  # Definition of events: event <event_id> { <source> <condition> }
7  event time_to_sleep {timer = 1mins}
8  event time_to_wakeup {timer = 29mins}
9  event danger {smoke = yes}
10  event safe {smoke = no}
11  # Definition of policies: from <state_id> to <state_id> when <event_id>
12  from hvac goto idle when time_to_sleep
13  from idle goto hvac when time_to_wakeup
14  from any goto fire when danger
15  from fire goto hvac when safe
16  start hvac                        # Initial state: start <state_id>
```

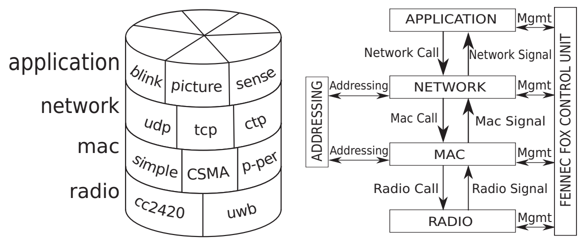
Figure 2. Sample Swift Fox Program

sensing, actuating, transmitting data. Events are defined by specifying a source of an event and a threshold value *x*, *e.g.,* a temperature sensor sensing more than *x* degrees, or a timer ending after *x* time units. Events are used to trigger transitions between network states. By combining the definitions of states and events, a programmer can specify various policies to control the dynamic reconfiguration of LPWN.

A sample program implementing a three-state network presented on Figure 1(a) is shown on Figure 2. Lines 3-5 define various network configurations, each specifying library modules supporting services provided by the layers of the network protocol stack. A library module implements functionality used by one of the layers of the network protocol stack, *i.e.,* application, protocol, or radio driver. A library module is written in nesC language as a generic component taking various number of parameters, whose values are specified in the Swift Fox program. For example, when a network is running in configuration *hvac* (lines 3), the application layer is executing module *temp* reporting an average of the last 3 temperature sensor samples. Network layer communication is established using the *ctpf* network library module supported with 16-bit addressing scheme from the *fixed* addressing library module. Lines 7-10 show definitions of events, each consisting of a name of an event library module and a threshold value. For example, event *time_to_sleep* takes place 1 minute after it is enabled (line 7). Based on configuration and event definitions, network reconfiguration policies are programmed, as shown on lines 12-15. For example, whenever the *danger* event occurs the network reconfigures to the *fire* state. The network starts at the *hvac* state.

A given program is processed by the Swift Fox compiler that links the code with the library modules used in the program and outputs nesC code. The output code implements Fennec Fox self-reconfigurable network protocol stack with mechanisms responsible for network reconfiguration, state synchronization, event detection and policy execution. Supported with the low-level operating system services provided by the TinyOS, the code is further compiled into a system image ready to be deployed.

Reconfigurable Network Protocol Stack. Fennec Fox provides a network communication protocol stack with four layers. Each layers is supported by a set of alternative modules, each implementing the full layer functionality while being optimized for a target performance metric, as shown on



(a) Four Layer Stack (b) Modules and Interfaces

Figure 3. Fennec Fox States

Figure 3(a). At runtime, Fennec Fox dynamically adapts the network to the given changes in the application scenarios by reconfiguring the combination of modules that are executed for each layer.

In order to be compatible with the Fennec Fox network protocol stack, a library module must provide and use the interfaces of the layer that it becomes part of. Specifically, a module provides an interface to the upper layer by implementing the functions whose definitions are part of that interface. A module, instead, uses the interface provided by a lower layer by calling functions defined by the interface and by implementing signal handlers that are sent at runtime from the lower-level modules. Figure 3(b) shows the top-down bottom-up message exchange interfaces between the layers of the network stack. The figure also depicts horizontal communication between the modules. Both network and MAC type library modules are supported with the same or two different addressing library modules, because Fennec Fox decouples communication problems from addressing. The Fennec Fox Control Unit uses management interface (*Mgmt*) to start and stop execution of modules across the layers of the network stack. The Control Unit is also responsible for executing network policies, keeping track of events, and synchronizing all network nodes to the same configuration, using Trickle-based state synchronization protocol.

2 Demonstration Description

We show a complete process of designing, programming, deploying, and executing event-driven self-reconfigurable LPWN. First, we present scenarios requiring different network communication characteristics. Second, we create a FSM model of a system consisting of two or more scenarios, with events detected through light and timers. Using Swift Fox we program the presented system model, compile it, and install it on the sensor motes. Finally, we create events triggering the network self-reconfiguration mechanisms.

Acknowledgments: This research is sponsored in part by the National Science Foundation (under Awards #: 0644202 and 0931870.)

3 References

[1] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *SenSys*, 2010.

[2] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys*, 2009.

[3] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *SenSys*, 2007.