Scriptable Sensor Network

Fajran Iman Rusadi
Universiteit van Amsterdam
Email: frusadi@science.uva.nl

ZhengZhangzheng
Universiteit van Amsterdam
Email: zzheng@science.uva.nl

***Abstract*—Each sensor node in the sensor networks is typically programmed for a certain task. Changing task means changing application that is run on the sensor node and this is difficult to be done. We provide a solution to make the sensor network scriptable that is also allows us to dynamically install new applications through the network. We will also show that our solution can be used as a foundation to build an active network.**

I. Introduction

A sensor network is a network that is composed of a collection of sensor node [1], a small device that has sensors to collect data, processor, and communication component. Sensor networks are used for several applications like measuring temperature, humidity, pressure, and other data that can be measured from environment.

Each sensor node in the sensor networks is typically programmed for a certain task. Changing task means changing application that is run on the sensor node. Application installation on the sensor network become something that is difficult to be done [4].

To handle this problem, we propose a solution to make the sensor network scriptable. Meaning that we can run scripts on the sensor node that also can be installed dynamically over the network. Hence, application installation become easier.

The application architecture that is used in our solution is flexible and also powerful to handle many kind of application. Not only applications that use the sensor node and the network, but also applications that control the network. We will show this by building an active network based on our solution. An active network is a network architecture that allows users to inject programs to the network nodes [5] that later on can do some computations on the messages flowing through them [2]. We call this as programmatically change the network behavior.

II. Background

Normally, a sensor network is used to do a certain task such as doing temperature measurement. This task is usually done by an application installed on each sensor node. When we want to make the sensor network do another task, we need to install a new application that does the new task.

Installing new application might be not an easy task [4]. Having to visit each sensor node and install new application one by one requires a lot of human resources and costs money. This cost become higher as more sensor networks deployed in the field.

To reduce this cost, we need some mechanism that allows us to dynamically install new applications on the sensor nodes. Dynamic here means we can install or remove applications over the network itself and therefore we don't have to individually visit the sensor node and install the new application.

Beside that, we might also need to change the network behavior. For example, to save more battery power, we need to use a better routing algorithm that pays attention to the battery power level. Another example is when we need to make segmentation in the network. We have to make sure that a message in one network segment is not broadcasted or forwarded to the another network segment.

This idea of changing network behavior leads to issue of creating active network. We want to control the network behavior programmatically by sending messages that can change the way the network works.

III. Lua SPOT

To answer the problem described in the previous section, we came up with idea to run one or more scripts on top of the sensor node. We first run our native application on the sensor node and provide it a basic capabilities to run script on the application. The application handles the installation and execution of scripts so we can dynamically install and execute scripts on the sensor node. We call our application Lua SPOT.

As implied in the name, Lua SPOT is able to run scripts written in Lua language. Lua is choosen because it is fairly simple yet powerful scripting language and it is designed as an embedded scripting language [3]. Lua needs a virtual machine to be run on and we use Kahlua [1], an open source Lua Virtual Machine that is written on Java langauge which can be run on top of Sun SPOT, the sensor node that we use.

We will describe Lua SPOT in detail in the following sections.

A. Software Architecture

First of all, we would like to give a general overview of the building blocks of Lua SPOT. As shown in Figure III-A, generally Lua SPOT is composed of three layers including the Sun SPOT layer where Lua SPOT runs on. This layer is where Lua SPOT gets executed. It also provides APIs to use the wireless network, access sensors and other input/output ports that can be used by any Sun SPOT application that run on top of it.

[1]http://code.google.com/p/kahlua/

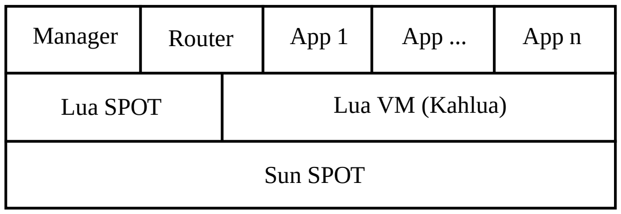

Fig. 1. Software Architecture

The middle layer is where Lua SPOT resides. We also put the Lua VM there since it will be part of Lua SPOT. This layer provides a script execution environment that handles scripts execution as well as installation and removal. APIs exposed by the Sun SPOT layer will also be exported to the scripts by creating function wrappers that can be called from Lua scripts above.

Lua scripts run on the top layer and they will be the application logic that controls the sensor node. The scripts will be run on the Lua virtual machine and can use APIs that are provided by the Lua SPOT to access functions that are provided by Sun SPOT and the Lua SPOT itself.

B. Service Provider

In Lua SPOT, we introduce a term Service Provider. We design the sensor network as a service provider that provides services which can be used by other entities.

Applications on the Lua SPOT will provide services. Functions inside each application can be called by other entities using some mechanism. Since the interaction between the functions and other entities is basically using function call, or remote function call to be precise, therefore an RPC like mechanism will be used to invoke a function inside an application.

Other entities that want to access a function inside a sensor node should send a message that represents an RPC. For the sake of simplicity, we design the RPC message as shown in Figure III-B.

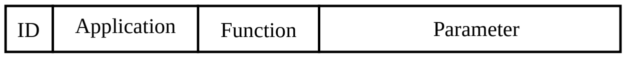

Fig. 2. RPC Message Format

It contains four fields and each field is separated by a single space. The first field will be the message identifier which distinguish our packet with any other packet. It contains a single character `c`, a short for ”call”.

The second and third field will be the application and function identifier. At the moment, we simply use the application and function name. The last field is the parameter that will be supplied to the function. Multiple parameters will be merged into one parameter and it is the function responsibility to parse the parameter into multiple parameters.

Based on this very basic idea of making the sensor network as a service provider, we can build many useful and powerful applications. For example, a routing function can be implemented as just another function. Messages that need to be routed will be inserted as a message of a function call to a routing function. In the last part of this section, we will show you an example of routing function that we make as part of basic pre-installed applications.

The routing function itself can be replaced or even we can install a new one that suits better with the requirements at a time. So, changing the behaviour of a network can be as simple as inserting a new application.

C. Application Execution

When the connection listener in the Lua SPOT receives a message, it will create a new thread and pass the message to a function called `dispatch()` from inside the thread. This function handles the initial message processing. It drops unwanted messages and extracts the application and function name as well as the parameter.

After knowing the application and function name, the `dispatch()` function will create a new Lua Virtual Machine and invoke the requested application and function. Each new Lua Virtual Machine contains the Lua standard library, the Sun SPOT and Lua SPOT libraries will be described later, the requested application code, and all other installed applications.

All installed applications will be available inside the Lua Virtual Machine. One application can call function on another application.

Since each function call is executed under a separate Lua Virtual Machine, this means the function call is stateless because any state will be destroyed once the function returns. One function execution and other function executions that happen at the same time can’t interfere each other.

However, a special APIs are provided by the Lua SPOT that allow the applications to share a global state. This will be described in the next section.

D. Sun SPOT and Lua SPOT APIs

Originally, the APIs that are available inside the Lua Virtual Machine is very limited. It only contains the Lua standard library so the scripts run on it are useless since they can only process something (the parameter) from the network but can’t do something to the network.

Therefore, a set of functions that are categorized into two APIs are provided: Sun SPOT and Lua SPOT APIs. The Sun SPOT API contains functions to access facilities provided by the Sun SPOT such as sensors and other input/output ports. The second API contains other functions that are required to make applications run on Lua SPOT more powerful, such as the global memory storage, synchronization, and also function to send message to the network.

E. Basic Applications

There are two basic applications in the Lua SPOT: Application Manager and Basic Router. The first application is responsible for adding, removing, and installing new application. The second application provides a routing function to route a message from one sensor node to other sensor node.

1) Application Manager: Installing a new application in Sun SPOT is just a matter of calling a function. A default application called Application Manager is reponsible to handle this kind of things. This application, which is named `manager`, is written in Java and will be the only application that is written in Java.

The application has two main functions: `install` and `remove`. The `remove` function need an application name that will be removed as the parameter. The `install` function needs more complex structure of the parameter since special care is needed when receiving new application data.

The size of message that can be transmitted to the network in one go is limited. This can be considered as the MTU in the regular network. Since the application size can be larger than the MTU, a data fragmentation is needed. Therefore, the parameter of the `install` function contains information about fragments. Figure III-E1 shows the message structure that is expected by the `install` function.

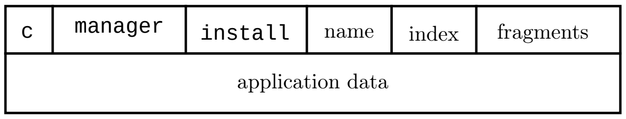

Fig. 3. Application Installation Packet Structure

The function requires four parameters: the application name, the fragment index, the total fragments, and the application data. When receiving new install request (identified by index 0), the application manager will initialize new application slot in the memory. The application data is appended to that slot until the last fragment arrives. The application manager will then activate the application so it is ready to be used.

2) Basic Router: When a sensor node want to send a message to another sensor node, there is a case when they are not within a range so the message should be transmitted through one or more intermediate sensor nodes. In order to do this, a special function need to be used so an intermediate sensor network can receive and forward such message. This functionality is called routing.

Lua SPOT contains an application that does routing. It receives a message and if the message is not sent to it, it will forward the message. The message will be hopped from one sensor node to the other until eventually it reaches the destination. If the routing application receives a message that is directed to it, it will remove the routing header and process the original message. The processing is done by calling the `dispatch()` function from inside application.

This routing application might be very basic, but it can already be used to forward a message from one node to another node that are not withing a range. This application is implemented as a Lua script and can be replaced if needed.

Just like the install function, routing function is implemented as another function that can be called. The name of application is `routing` and the function that will do routing is `route`. The function expects several parameters as shown in Figure III-E2.

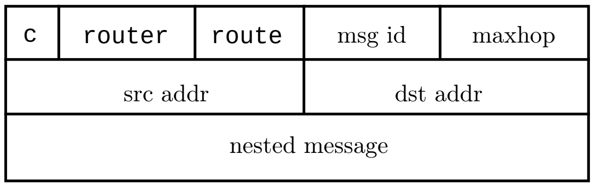

Fig. 4. Router Packet Structure

The message contains message id that will be used to avoid processing duplicate messages, it also has a maximum hop number to limit the message distribution. Source and destrination addresses are obviusly used to state the sender and receiver. The last part of the message is another message that is embedded in this message. As mentioned earlier, that nested message will be sent to the `dispatch()` function so it can be processed just like any other message.

IV. Lua SPOT and Active Network

One of the key concept of Active Network is an ability to change network behavior programmatically [5]. A message that is sent to a node may contain a program that will be executed on the node. This program will then run and control the traffic that passes through the node. Furthermore, a message might be an active ”capsule”, a small program that gets executed in every router/switch that it traverses.

In this section, we would like to show that Lua SPOT can be used as a foundation in building an active network. We will try to design a Lua SPOT application that can show features of active network.

A. Interacting with Lua SPOT

The basic and only interaction with Lua SPOT is by sending an RPC message. Lua SPOT does nothing but accepting an RPC message and dispatch the message to the corresponding application and function. Therefore, all features that are expected from a network device are implemented as functions inside applications or, as we mention earlier, services.

Lua SPOT has a very main function that acts as gateway of command execution, the `dispatch()` function. This function is called when Lua SPOT receives a new message. This function can also be called from inside an application which opens possibility of executing any message that is created from inside the application, including a message that is taken from the parameter of the function. That is, a message that is passed as a parameter through the `dispatch()` function that might come from other network elements.

At the current implementation of Lua SPOT, the created Lua Virtual Machines will contains all installed applications. A function that is invoked can access all other available functions. In this way, any new application that is installed in the sensor node will enrich the functionality of the sensor node, and therefore the sensor network. The installed application can have functions that are not intended to be called remotely, but solely for making the sensor node richer.

B. Creating Active Network

Based on Lua SPOT approach described in the previous section, we will design an application that can support two scenarios that are expected from an Active Network, as discussed in [5]. An ability to execute a program embedded in the message and let the program to control the traffic; and a support to send ”capsule” to the network, a message that contains an embedded program which will be executed on the receiving nodes and forwarded to other nodes.

The first scenario can be achieved at least in two ways. The first one is by replacing the router application by another application that is smarter in processing the incoming messages. If this routing function is a standard routing function that is used to carry all other messages, replacing it means changing the way the network works.

Another way to achieve the first scenario is by inserting a new routing function. Any other messages that we want to transmit can be carried using that routing function. Therefore, if we have messages that need to be handled (routed) differently, we can use this new routing function to carry them.

A ”capsule” message can be handled by Lua SPOT by having a function that does two things: routing and dispatching. When the function receives a message, it will execute the embedded message by passing it to the `dispatch()` function and also forward the message to the other nodes. In this way, a message will get executed in every intermediate nodes that it traverses.

V. Experiments

To test Lua SPOT, we made two experiments. The first experiment is about testing the installation function. If this experiment is successful, then we can answer the problem described in the earlier section: we need to be able to dynamically deploy new applications on the sensor nodes.

In the second experiment, we would like to show a collective behavior of sensor nodes. We inserted an application that will exchange a measurement data to other sensor nodes. Then, each sensor node will independently calculate the data to build a routing path. Another command later on will be sent to the sensor network and they should be able to pass the command one by one along the path they made earlier.

A. Application Installation

In this experiment, we tried to send a installation command message. We installed several applications ranging from small application to a relatively large application, in terms of file size.

B. Collective Behaviour of Sensor Nodes

This experiment has a goal to show a collective behaviour of sensor networks. We were trying to make a routing path that is determined by a certain measurement, we use the value of accelerometer provided by Sun SPOT, taken by each sensor node.

The routing path calculation is done in a distributed fashion since there is no single sensor node that acts as a central controller. However, we still need a sensor node that will send a message to all other sensor networks that trigger the exchange, so all sensor nodes can start calculating the routing path. After that, we send a message to all sensor networks that should be responded by a sensor node that has a smallest measurement value. It then send a message to its next sensor node in the routing path calculated earlier.

VI. Conclusion

In conclusion, we have successfully created an application on sensor network that is able to run scripts. Beside running the pre-installed scripts, new scripts can also be installed in this application. This opens posibility to add and run new scripts after the deployment of sensor networks without having to visit each sensor node individually.

A. Future Works

The current implementation of Lua SPOT still doesn’t export all functions of Sun SPOT to the Lua VM. It also doesn’t pay attention to the memory usage of Lua VM as well as the applications that will be run on it. The message format is also not very efficient and not really binary message friendly.

In addition to fix the limitation at the current implementation, what we would like to see on Lua SPOT is some new improvements. The `dispatch()` function currently work synhronously. Means it will block the execution when it is called. An asynrhonous version of it might be makes the Lua SPOT more powerfull and can do something like invoking a function several times in relatively parallel.

To make script development easier, an SDK and simulator for Lua script might also a good idea. The developers doesn’t need Sun SPOT if they have something that can be used to test the scripts before actually deploying them to the Lua SPOT.

References

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] D. Tennenhouse *et al.* A survey of active network research. *ieeecom*, 35(1):80–86, 1997.

[3] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua — an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[4] Rene Müller, Gustavo Alonso, and Donald Kossmann. A virtual machine for sensor networks. In *Proceedings of the Second European Systems Conference (EuroSys2002)*, Lisbon, Portugal, March 2007. ACM SIGOPS.

[5] David L. Tennenhouse and David Wetherall. Towards an active network architecture. *Computer Communication Review*, 37(5):81–94, 2007.