Implementation of GUI for OpenThread

Jitendra Gopaluni
Computer Engineering
University of Houston Clear Lake
Houston, USA
jitendra0310@gmail.com

Ishaq Unwala Ph.D.
Computer Engineering
University of Houston Clear Lake
Houston, USA
Unwala@uhcl.edu

Jiang Lu Ph.D.
Computer Engineering
Unversity of Houston Clear Lake
Houston, USA
LuJ@uhcl.edu

Xiaokun Yang
Computer Engineering
University of Houston Clear Lake
Houston, USA
YangXia@uhcl.edu

***Abstract*— Thread is an IPV6 based networking protocol for Internet-of-Things (IoT) that uses 6LoWPAN, IEEE 802.15.4 wireless at 2450MHz. OpenThread is an open source implementation of Thread Protocol released by Nest Labs that supports the Thread 1.1.1 specifications and currently used on Linux using command-line interface. This research focuses on building a Graphical User Interface (GUI) using TCL/TK for OpenThread. GUI helps researchers to manage and visualize Thread Protocol and its operations easier.**

***Keywords*— IoT, IEEE 802.15.4, TCL/TK, Thread Protocol**

I. Introduction

Internet-of-Things (IoT) has seen a great growth and remarkable development the last few years. It is a major challenge, and also an opportunity, for home and industrial automation. Home automation is an excellent example of application of the IoT.

IoT uses the idea of sensor and actuator devices connected to form a Private Area Network (PAN). The access for these devices to the Internet is through a gateway (border) router. All devices (nodes) on the PAN are connected with a wireless connection (edge). The topology of a PAN depends on the protocol, star, peer-to-peer and mesh topologies are possible candidates. The border router allows cloud computing access for the devices in the PAN.

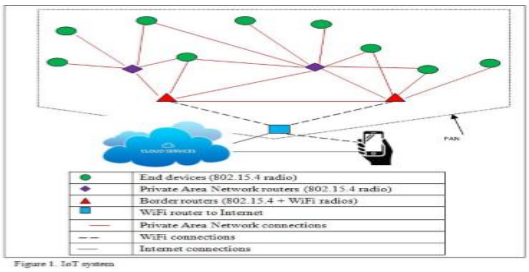

Figure 1. IoT system

II. Thread Protocol

The existing home automation technologies and products in the market like Z-Wave [1] or EnOcean [2] do not meet all of the requirements of the low power, resilience, IP-based, security and friendly use. Thread protocol [4] attempts to achieve these requirements of low power by using wireless connections based on IEEE 802.15.4 operating at 2450 MHz It uses IPv6 addressing compressed with 6LoWPAN. It uses mesh topology to increase resilience. Thread PAN can have multiple gateways to Internet, thus removing a single point of failure. Thread is an open protocol as it is based on IEEE standards and IETF RFCs.

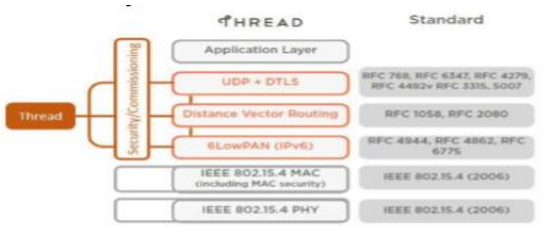

Figure 2. Thread protocol stack of IOT systems

III. OpenThread

Nest Labs released OpenThread in May 2016, which is an open source implementation based on Thread 1.1.1 specification of the Thread Networking protocol. Along with Nest Labs, ARM, Atmel, Dialog semiconductors, Qualcomm Technologies and Texas Instruments are also contributing to the ongoing development of OpenThread. OpenThread implements all Thread networking layers including:

- IEEE 802.15.4 with MAC security.
- IPv6 and 6LoWPAN.
- Mesh Link Establishment and Mesh Routing.
- Key management.
- Definitions in code of specific roles in Thread including:
  - Leader.
  - Router.
  - End Device.
  - The Border router.
- UDP packet compression.
- A CoAP implementation.

OpenThread is portable, OS and platform, agnostic with a radio abstraction layer, written in C++. OpenThread depends on a platform layer, basically a Hardware Abstraction Layer (HAL), and when the layer is implemented, OpenThread can run on most microcontroller or 802.15.4 SoCs with advantage of small memory print. OpenThread is available at GitHub repository.

978-1-7281-1360-9/18/$31.00 ©2018 IEEE
DOI 10.1109/CSCI46756.2018.00196

Authorized licensed use limited to: Univ Politecnica de Madrid. Downloaded on January 30,2021 at 10:46:57 UTC from IEEE Xplore. Restrictions apply.

A. Network Implementaion

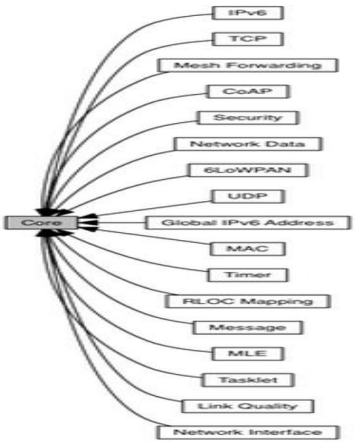

Figure3 Overview of OpenThread modules. [5]

IPv6 includes definitions for the IPv6 network layer. IPV6 module (Figure.4) defines the ICMPv6 implementation the network interfaces, the multicast protocol and the IPv6 implementation itself.

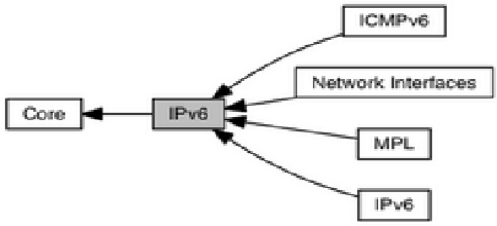

Figure 4. IPv6 module details[5]

B. Routing Implementation

OpenThread execute MLE to spread the Routing table data and RIPng to process data and keep up directing tables. The executed MLE module (Figure. 5) Relies upon Core and executes MLE usefulness required for the Thread Router and Leader jobs. The Type Length Value (TLV) module incorporates definitions for creating and handling MLE TLVs.

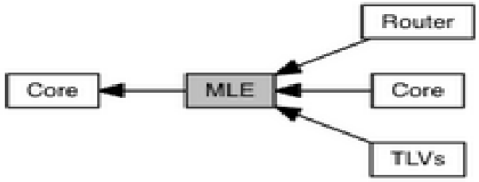

Figure 5. MLE Module details. [5]

C. DTLS Implementation

DTLS class (Figure. 6) is part of the Mesh Cop class and defines all the related messages to implement the DTLS functionalities in OpenThread stack.

D. CoAP Management protocol implemetation

While the Thread convention utilizes CoAP for its control informing, the CoAP server examples inside a Thread arrange are devoted to Thread's control informing (i.e. Thread 's CoAP server utilizes an unexpected port in comparison to the default CoAP port).

In the meantime, OpenThread exposes a CoAP API that enables application to utilize the equivalent CoAP execution to send/get CoAP messages. In the event that you need to test that out, you should set up a CoAP server, include an asset, and execute the suitable higher-layer rationale to send and process CoAP messages. The UDP port for Thread-particular is 61631.

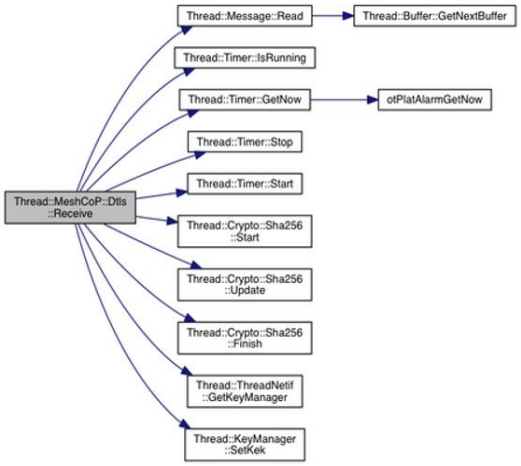

Figure 6. DTLS module details. [5]

At present the CoAP server question in OpenThread just actualizes the subset of the convention that is essential for Thread Commissioning. OpenThread CoAP API is incapacitated as a matter of course. To empower that include, you should add `-empower application-CoAP` to the `./configure` choices

Joiner Entrust CoAP message and it will be anchored on MAC with Key ID Mode 0, utilizing single-utilize KEK. Be that as it may, despite everything it is a confirmable message, and CoAP may need to retransmit it. CoAP messages are transmitted over anchored DTLS association.

E. CLI commands

The OpenThread CLI exposes configure and management APIs via a command line interface. Use the CLI to play with OpenThread, which can also be used with additional application code.

Some of the CLI available commands are:

- autostart
- Commissioner
- factoryreset
- ifconfig up
- ipaddr
- joiner
- panid

Authorized licensed use limited to: Univ Politecnica de Madrid. Downloaded on January 30,2021 at 10:46:57 UTC from IEEE Xplore. Restrictions apply.

- parent
- ping.
- reset.
- rloc16
- router
- state

IV. GUI FOR OPENTHREAD

Using OpenThread on CLI to implement a network, each node is configure using a command prompt. Every time a node is configure using a command prompt, which complicates the work of the developer as well as the network with high number of nodes.

A. GUI for OpenThread

A GUI is implemented for OpenThread using TCL/TK. TCL/TK enables building a GUI natively in TCL. OpenThread having a GUI will make working with OpenThread much easier, the use of basic commands need not be memorized and additionally the user does not need to know any programming languages. The GUI uses the OpenThread CLI commands and creates the nodes and supports the CLI commands

B. GUI workings

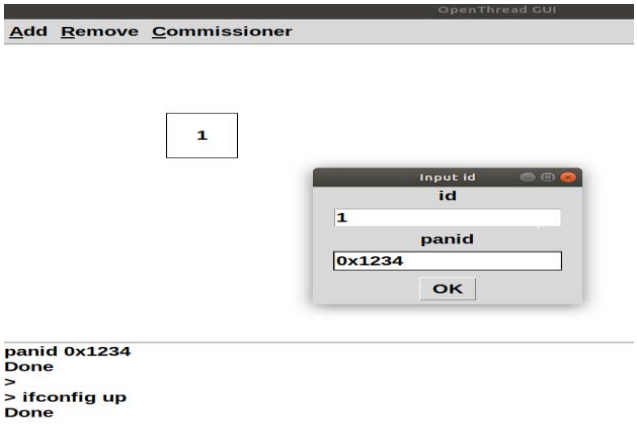

Figure 7. Creating a node.

In Figure 7, we create a node by an ID and panid. With these inputs from the user, a node is created with an ID as shown. Similarly, with the user defined parameters a second node is created with an ID '2' in Figure 8.

In Figure 9, the state of node 2 is shown. Once connected, the state of the node 2 is showed as router. Hence, we have configured node 1 to be a leader and node 2 is the router connected to the leader.

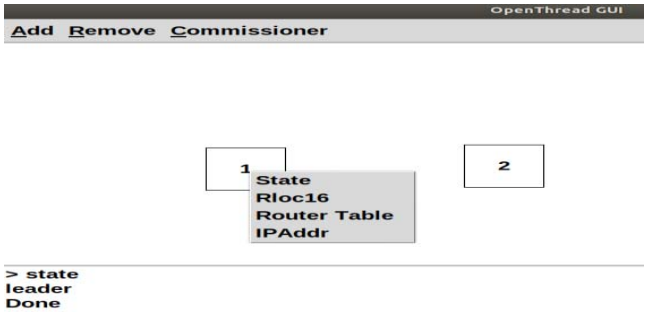

Figure 8. Node 1 State

The node with id '1' is configure to be a leader with the panid 0x1234. And Figure 8.1 gives the state of the router. It is seen that, the node 1 is leader. Similarly, node 2 is created with similar panid as that of the node 1. Since, having the same id, node 2 is connected to node 1.

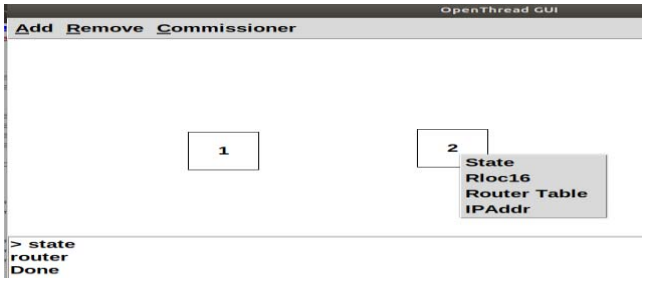

Figure 9. Node State

Figure 10 and Figure 11 proves the connection between the leader and the router. In Figure 10 we get the Rloc16 which is the 16bytes EID of the router node. And in the Figure 11, we get the router table of node 1 i.e. leader node.

In Figure 12, we have disabled the leader router using "stop" from "remove".

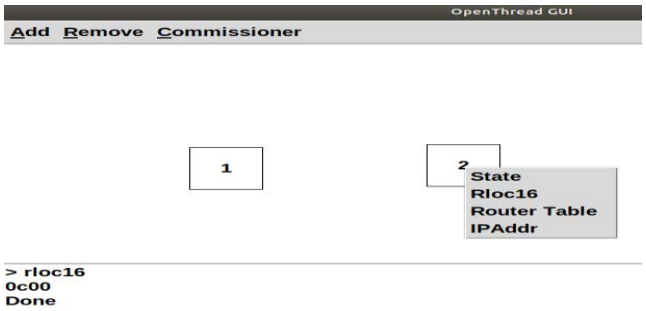

Figure 10. Finding the rloc16 of node 2

The router table of the leader has the 16bytes EID of the router node. It is seen from Figure 10; the EID value is 0x0C00 and the same value is in the router table of the leader router as show in Figure 11.

Authorized licensed use limited to: Univ Politecnica de Madrid. Downloaded on January 30,2021 at 10:46:57 UTC from IEEE Xplore. Restrictions apply.

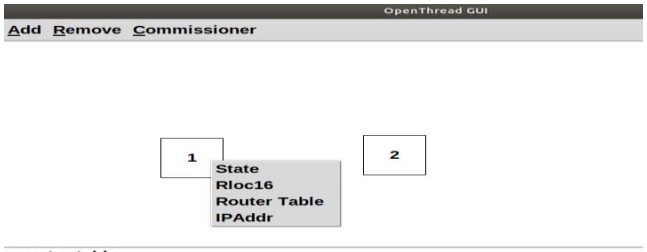

Figure 11. Testing the connection

Hence, we prove that the leader and the router nodes are connected to each other.

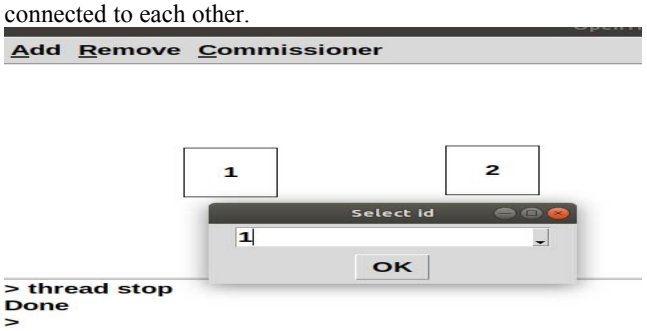

Figure 12. Disabling the leader node

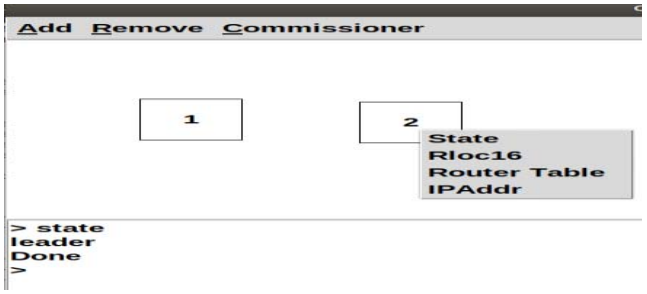

Figure 13. State of node 2

In Figure 12, the state of the leader, i.e. node 1 is disabled and according to thread protocol, when one leader is disabled, the next available router becomes Leader and a REED, becomes a router, this ensuring no point of failure.

In Figure 13, it is observed that the node 2, which was a router when the leader was active became a leader when the leader was in a disabled state.

In Figure 14, and Figure 15, we demonstrate the commissioner and joiner functions in the thread protocol. We created 2 nodes, where node 1 is the leader and node 2 is the joiner. Node 1 is configure to be a commissioner. In Figure 14, we assigned the leader with a panid and a user defined key i.e. “joinme”.

In Figure 15, we confugured the node 2 to be joiner by selecting the node id and giving the joiner key i.e “joinme”. And in the bottow of the screen, there is a confirmation which shows that the connection was success and the joiner joined the network.

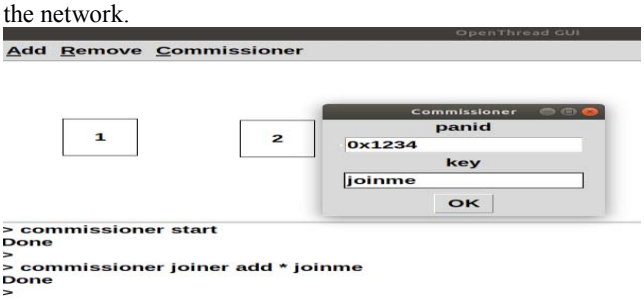

Figure 14. commissioner

This shows how OpenThread works on the CLI command Line Interface level. But instead of having a command prompt for each node, this GUI will make the reasearch involving OpenThread much easier to work with.

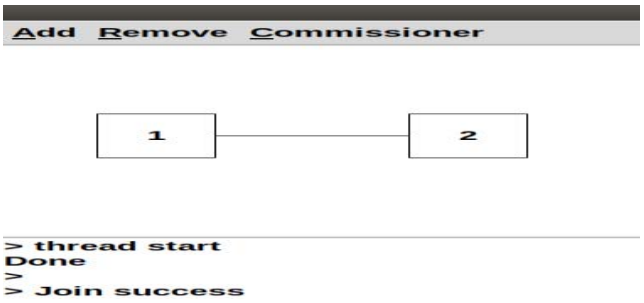

Figure 15 joiner command

V. Conclusion

Thus, having a GUI, developer can have any number of nodes on a single screen and the user doesn’t need to know any of the commands and languages, making things simple and visualizing OpenThread easy.

References

[1] Z-Wave products commercial website http://www.z-wave.com

[2] EnOcean commercial website https://www.enocean.com

[3] Thread website: https://threadgroup.org

[4] OpenThreadGithubwebsite:https://github.com/openthread/openthread

[5] Humberto Gonzalez Gonzalez , “Study of the Thread Protocol for Home automation Thread.”

[6] Ishaq Unwala, Z. Taqvi and J. Lu, "Thread: An IoT Protocol," 2018 IEEE Green Technologies Conference (GreenTech), Austin, TX, 2018, pp. 161-167. doi: 10.1109/GreenTech.2018.00037

[7] Ishaq Unwala, Jiang Lu, “IoT Protocols : Z-Wave and Thread”, November 2017 Volume 3 Issue 11 , International Journal on Future Revolution in Computer Science & Communication Engineering (IJFRSCE), PP: 355 – 359

[8] https://codelabs.developers.google.com/codelabs/openthread-simulation/#3

[9] Ishaq Unwala, Z. Taqvi and J. Lu, "IoT Security: ZWave and Thread," 2018 IEEE Green Technologies Conference (GreenTech), Austin, TX, 2018, pp. 176-182. doi: 10.1109/GreenTech.2018.00040

Authorized licensed use limited to: Univ Politecnica de Madrid. Downloaded on January 30,2021 at 10:46:57 UTC from IEEE Xplore. Restrictions apply.