International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 2, March - April 2017 ISSN 2278-6856

Role of COOJA Simulator in IoT

B. Sobhan babu[1], P. Lakshmi Padmaja[2], T. Ramanjaneyulu[3], I. Lakshmi Narayana[4], K.Srikanth[5]

[1,2,3,4,5]Gudlavalleru Engineering College, Gudlavalleru. Krishna District, Andhra Pradesh, India.

Abstract

The Internet of Things (IoT) is the network of physical objects or things embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data. The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure. The IoT is enabled by the latest developments in RFID, smart sensors, communication technologies, and Internet protocols. Cooja Simulator is a network simulator specifically designed for Wireless Sensor Networks. A simulated Contiki Mote in COOJA [1] is an actual compiled and executing Contiki system. The system is controlled and analyzed by COOJA. This is performed by compiling Contiki for the native platform as a shared library, and loading the library into Java using Java Native Interfaces (JNI). Several different Contiki libraries can be compiled and loaded in the same COOJA simulation, representing different kinds of sensor nodes (heterogeneous networks). COOJA controls and analyzes a Contiki system via a few functions. For instance, the simulator informs the Contiki system to handle an event, or fetches the entire Contiki system memory for analysis. This approach gives the simulator full control of simulated systems. Unfortunately, using JNI also has some annoying side-effects. The most significant is the dependency on external tools such as compilers and linkers and their run-time arguments. COOJA was originally developed for Cygwin/Windows and Linux platform, but has later been ported to MacOS. Java version 1.6 or later is required to run COOJA. We recommend using the latest version from Sun. In addition, the build tool ant is also required for building COOJA [12]. Contiki is an open source operating system for the Internet of Things. Contiki connects tiny low-cost, low-power microcontrollers to the Internet. Contiki is a powerful toolbox for building complex wireless systems.

Keywords: Contiki, COOJA, JNI, RFID.

1. INTRODUCTION

Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices. Extant uses for Contiki include systems for street lighting, sound monitoring for smart cities, radiation monitoring, and alarms. It is open-source software released under a BSD license. Contiki provides multitasking and a built-in Internet Protocol Suite (TCP/IP stack), yet needs only about 10 kilobytes of random-access memory (RAM) and 30 kilobytes of read-only memory (ROM). A full system, including a graphical user interface, needs about 30 kilobytes of RAM. Contiki is designed to run on types of hardware devices that are severely constrained in memory, power, processing power, and communication bandwidth. A typical Contiki system has memory on the order of kilobytes, a power budget on the order of milli-watts, processing speed measured in megahertz, and communication bandwidth on the order of hundreds of kilobits/second. Such systems include many types of embedded systems, and old 8-bit computers. Contiki is an open source operating system for the Internet of Things. Contiki connects tiny low-cost, low-power microcontrollers to the Internet. Contiki is a powerful toolbox for building complex wireless systems. Contiki provides powerful low-power Internet communication. Contiki supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6lowpan, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated. The development is easy and fast and also Contiki applications are written in standard C, with the Cooja simulator Contiki networks can be emulated before burned into hardware, and Instant Contiki provides an entire development environment in a single download. Contiki is open source software: Contiki can be freely used both in commercial and non-commercial systems and the full source code is available. The following figure shows the Cooja in Contiki network simulator.

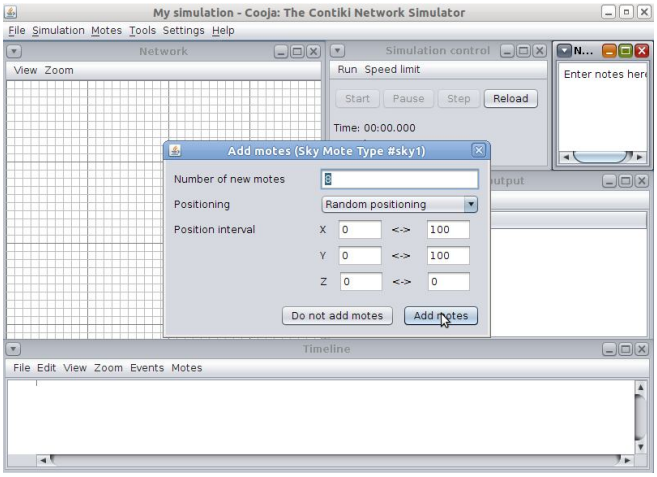

Fig1:COOJA in Contiki network simulator

The Internet of Things (IoT) is all about interconnecting of embedded devices to form a local mesh network or connect directly to the internet. The existing networking stacks are not optimal for IoT devices due to constraints like low power (and in most cases we don't have human intervention to replace the battery) and low processing power. For this type of requirements, the IETF (Internet Engineering Task Force) has drafted a few protocols to accommodate the new network requirements. To test and debug these new protocols we need to not only consider the functionality of the protocol but also test the properties of the device like battery performance and memory usage. For such requirements the traditional network simulators

won't suffice, so we have simulators especially for these devices. One such simulator for IoT devices is Cooja simulator. In this post, I would like to demonstrate how to use cooja as a network test bed for IoT devices, and integrate with the AWS IoT platform to communicate between two different mesh networks.

2. STACKS OF COOJA

COOJA has two stacks one is uIP[6] and another one is Rime. TCP/IP stacks are the global enablers of world-wide inter-connection and communication of devices and computers. A wide range of stack implementations is available today: from full-scale stacks in Windows and Linux to small-scale stacks for embedded [9] systems like lightweight IP (lwIP) [1] and even smaller stacks like micro IP (uIP) [1] for resource-constrained devices with 8-bit microcontrollers. Using IP (v6) [2] as the bridging protocol facilitates a seamless interconnection of both resource-constrained and non-constrained devices over the Internet (condensed under the label Internet of Things).

The layers of Rime are designed to be extremely simple, both in terms of interface and implementation. Each layer adds its own header to outgoing messages. Because Rime layers are simple, individual headers are very small; typically a few bytes each. The thin layers in Rime enable code reuse within the stack. For example, reliable communication is not implemented in a single layer but divided into two layers, one that implements acknowledgments and sequencing, and one that resends messages until the upper layer tells it to stop. We call the latter layer stubborn. A stubborn layer is not only used by reliable protocols but also by protocols that send periodic messages such as neighbor maintenance for routing protocols and repeated transmission of messages in Rime's network flooding layer The lowest level primitive in Rime is anonymous best- effort broadcast, abc. The abc layer provides a 16-bit channel abstraction but no node addressing; it is added by upper layers. The identified sender best-effort broadcast, ibc, adds a sender identity header field and the unicast abstraction, uc, adds a receiver header field. An underlying MAC or link layer may choose to implement parts of the Rime stack, such as the abc, ibc, or uc layers, in hardware or firmware.

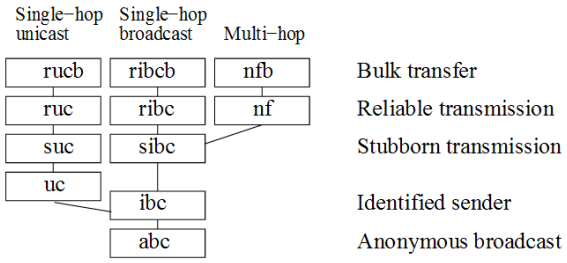

Fig:2 Rime Stack

2.1 SHIFTING THE BURDEN FROM APPLICATIONS TO THE SYSTEM CORE

One of the basic ideas of Rime is to shift the burden, in terms of memory footprint, from protocol implementations to Rime. By making Rime part of Contiki's system core, which is always present in memory, loadable programs are made smaller. Consequently, the energy consumption for program loading [2] is reduced.

2.2 BUFFER MANAGEMENT

To reduce memory footprint Rime uses a single buffer for both incoming and outgoing packets similar to uIP [1]. Layers that need to queue data, e.g. a stubborn protocol or a MAC layer, copy the data to dynamically allocated queue buffers.

Cooja can also useful to emulate RPL and LIBP network protocols. RPL (Routing Protocol for LLNs)[8] is a direct result of the Internet Engineering Task Force (IETF) which recognized the need to form a standardized IPv6 based routing solution for LLNs. The IETF formalized a working group specific for this problem called ROLL (Routing over Low power and Lossy). The direct outcome of this group was RPL. RPL is a Distance Vector IPV6 routing protocol for LLNs that specifies how to build a Destination Oriented Directed Acyclic Graph using an objective function and a set of metrics and constraints. RPL basically builds a logical communications graph over a physical network that conforms to satisfying a set of objectives and conforms to a set of constraints which can be set by a network administrator. The graph building process is initiated at the root node; multiple roots can exist in the same network. The roots start advertising the information about the graph using messages outlined in its RFC and other literature. LIBP, known as the Least Interference Beaconing Protocol [10], is the implementation of the Least Interference Beaconing Algorithm, LIBA. LIBP extends the beaconing process widely used by collection protocols with load balancing to improve the Ubiquitous Sensor Network (USN) energy efficiency [4]. The process involving the least interference paradigm allows the selection of a parent node that has the smallest number of children. This is a point of least traffic flow interference. The parent selection model chooses the first parent node heard from, whereby the sensor nodes hear from a set of neighbours and select the least burdened (in number of children) as the parent node. RPL has two objective functions one is objective function ETX and another one is Objective Function Zero. The ETX objective function is a widely popular link metric in the field of WSN. It is a link metric that in some way encompasses link congestion and link latency. ETX is simply defined as the expected number of transmissions required to successfully transmit and acknowledge a packet on a wireless link. The objective function Zero is a relatively new objective function proposed by the IETF. In comparison to ETX it is not highly established since ETX is considered a mature link metric in the field of WSNs.

3. NETWORK PROTOCOLS OF COOJA

3.1 LIBP (LEAST INTERFERENCE BEACONING PROTOCOL)

LIBP, known as the Least Interference Beaconing Protocol, is the implementation of the Least Interference

Beaconing Algorithm, LIBA. LIBP extends the beaconing process widely used by collection protocols with load balancing to improve the Ubiquitous Sensor Network (USN) [13] energy efficiency. The process involving the least interference paradigm allows the selection of a parent node that has the smallest number of children. This is a point of least traffic flow interference. The parent selection model chooses the first parent node heard from, whereby the sensor nodes hear from a set of neighbours and select the least burdened (in number of children) as the parent node. LIBA is an implementation of the LIBA algorithm. This routing protocol, like CTP, uses a beaconing process initiated by the source node. When the process is initiated nodes incidents to the sink node will be the first to recognize that a sink node is within one hop distance. This process is then initiated by these nodes to their neighbours and this process is repeated thereafter. This results in a network where each node is aware of its neighbours. The least interference paradigm is integrated into the process by which nodes select parent nodes which have the smallest number of children, which is the parent of least traffic flow interference. This configuration is especially powerful in the situation where sensors are periodically sensing information (which is a very popular sensor use case). LIBP basically aims to provide a way to balance traffic flow in such a way that it results in energy efficiency by having a network where nodes support less traffic.

3.2 CTP (Collection Tree Protocol):

CTP is a routing protocol which extends the Trickle Algorithm. It does so because the assumption can be made that data aggregation is one of the primary goals of a WSN. CTP promises to be reliable, efficient, robust, and hardware independent. CTP relies on data packets to validate the routing topology and loop detection. This routing protocol also utilizes adaptive beaconing to dynamically setup and adapt to network changes. Every node implementing CTP maintains an estimate of the cost of its route to a collection point. This metric is typically called Expected transmission (ETX).

3.3 Routing Protocol for LLNs (RPL)

RPL is a direct result of the IETF which recognized the need to form standardized IPV6 based routing solution for LLNs. The IETF formalized a working group specific for this problem called ROLL (Routing over Low Power and Lossy). The direct outcome of this workgroup was RPL.

4. Features of COOJA

4.1 Memory Allocation

Contiki is designed for tiny systems, having only a few kilobytes of memory available. Contiki is therefore highly memory efficient and provides a set of mechanisms for memory allocation: memory block allocation memb, a managed memory allocator mmem, as well as the standard C memory allocator malloc.

4.2 Full IP Networking

Contiki provides a full IP network stack, with standard IP protocols such as UDP, TCP, and HTTP, in addition to the new low-power standards like 6lowpan, RPL, and CoAP. The Contiki IPv6 stack, developed by and contributed to Contiki by Cisco, is fully certified under the IPv6 Ready Logo program.

4.3 Power Awareness

Contiki is designed to operate in extremely low-power systems: systems that may need to run for years on a pair of AA batteries. To assist the development of low-power systems, Contiki provides mechanisms for estimating the system power consumption and for understanding where the power was spent.

4.4 6lowpan, RPL, CoAP

Contiki supports the recently standardized IETF protocols for low-power IPv6 [5], [9] networking, including the 6lowpan adaptation layer, the RPL IPv6 [4] multi-hop routing protocol, and the CoAP RESTful application-layer protocol.

4.7 Dynamic Module Loading

Contiki supports dynamic loading and linking of modules at run-time. This is useful in applications in which the behavior is intended to be changed after deployment. The Contiki module loader [15] can load, relocate, and link standard ELF files that can optionally be stripped off their debugging symbols to keep their size down.

4.8 Sleepy Routers

In wireless networks, nodes may need to relay messages from others to reach their destination. With Contiki, even relay nodes, so-called routers, can be battery-operated thanks to the Contiki MAC radio duty cycling mechanism which allows them to sleep between each relayed message. Some call this sleeping router, we call it sleepy routers.

4.9 Hardware Platforms

Contiki runs on a wide range of tiny platforms, ranging from 8051-powered systems-on-a-chip through the MSP430 and the AVR to a variety of ARM devices. There are also a number of more exotic platforms thrown in there for good measure. Read more about Contiki hardware platforms.

4.10 Proto-threads

To save memory but provide a nice control flow in the code, Contiki uses a mechanism called proto-threads. Proto-threads are a mixture of the event-driven and the multi-threaded programming mechanisms. With proto-threads [2], event-handlers can be made to block, waiting for events to occur.

4.11 COFFEE FLASH FILE SYSTEM

For devices that have an external flash memory chip, Contiki provides a lightweight flash file system, called Coffee. With Coffee, application programs can open, close, read from, write to, and append to files on the external flash, without having to worry about flash sectors needing to be erased before writing or flash wear-leveling. The performance of Coffee is within 95% of the raw throughput of the flash memory.

4.12 THE CONTIKI SHELL

Contiki provides an optional command-line shell with a set of commands that are useful during development and debugging of Contiki systems. With Unix-style pipelines, shell commands can be combined in powerful ways. Applications can define their own shell commands that work together with existing commands.

5 .CONCLUSION

The resulting simulator [11, 12] works well for the intended usage. It helps new users to quickly and easily start up a simulation, and is very useful during development and test phases. It supports heterogeneous networks, concerning both simulated hardware and software. Larger-scale behaviour protocols and algorithms can be observed by using the basic set of plugins or by easily extending them. However, due to its extendibility it is not extremely efficient. It is better suitable for developing IoT applications and simulates those applications with the help of Cooja simulator.

References

[1] A. Dunkels, B. Gr¨onvall, and T. Voigt. Contiki - a Light weight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop onEmbedded Networked Sensors*, Tampa, Florida, USA, Nov.2004.

[2] Dunkels, Adam; Schmidt, Oliver; Voigt, Thiemo; Ali, Muneeb (November 2006), "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems", Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys), Boulder, CO, USA Dunkels, A.; Schmidt, O.; Voigt, T.; Ali, M. (2006). "Protothreads". Proceedings of the 4th international conference on Embedded networked sensor systems - Sen Sys '06. p. 29.

[3] M. Karir. Atemu - sensor network emulator / simulator / debugger. Technical report, Center for Satellite and Communication Networks, Univ. of Maryland, 2003.

[4] Alan, A., & Pritsker, B. (n.d.). Why Simulation Works. Proceedings of the 1989 Winter Simulation Conference.

[5] J. Hui and E. Culler, "Extending IP to Low-Power, Wireless Personal Area Networks," Internet Computing, IEEE, vol. 12, no. 4, pp. 37–45, August 2008.

[6] Contiki Doxy gen Documentation, "uIP TCP Throughput Booster Hack," [online]. http://contiki.sourceforge.net/docs/2.6/a01696.html

[7] R. Klauck and M. Kirsche, "Bonjour Contiki: A Case study of a DNS-Based Discovery Service for the Internet of Things," in Proceedings of the 11th International IEEE Conference on Ad-Hoc Networks and Wireless (ADHOC-NOW 2012), ser. Lecture Notes in Computer Science (LNCS), vol. 7363. Springer, July 2012, pp. 316 – 329. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31638-8 24.

[8] Contiki, "Contiki: The Open Source Operating System for the Internet of Things," 2015.

[9] M. Durvy, J. Abeill´e, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making Sensor Networks IPv6 Ready," in Proc. of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys 2008), poster session, November 2008.

[10] R. Bless and M. Doll, "Integration of the FreeBSD TCP/IP-Stack into the Discrete Event Simulator OMNet++," in Proceedings of the 36th Conference on Winter Simulation (WSC), 2004, pp. 1556–1561.

[11] F. Oesterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross- Level Sensor Network Simulation with COOJA," in Proc. of the 1st IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), Nov. 2006.

[12] C. A. Boano, K. Römer, F. Österlind, and T. Voigt. Demo Abstract: Realistic Simulation of Radio Interference in COOJA. In Proceedings of the uropean Conference on Wireless Sensor Networks (EWSN), Bonn, Germany, 2011.

[13] M. Karir. atemu - Sensor Network Emulator / Simulator / Debugger. Technical report, Center for Satellite and Communication Networks, Univ. of Maryland, 2003.

[14] D.Watson andM. Nesterenko. Mule: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks. Workshop on Sensor and Actor Network Protocols and Applications, 2004.

Author Profile

B.SobhanBabu[1] M.Tech, and working as Assistant Professor in Department of Information Technology, Gudlavalleru Engineering College, Gudlavalleru, AndhraPradesh.

P. Lakshmi Padmaja[2] M.Tech, and working as Assistant Professor in Department of Information Technology, Gudlavalleru Engineering College, Gudlavalleru, AndhraPradesh.

T.Ramanjaneyulu[3] M.Tech, working as Assistant Professor in Department of Information Technology, Gudlavalleru Engineering College, Gudlavalleru, Andhra Pradesh.

I. Lakshmi Narayana[4] M.Tech, working as Assistant Professor in Department of Information Technology, Gudlavalleru Engineering College, Gudlavalleru, AndhraPradesh.

K.Srikanth[5] M.Tech, working as Assistant Professor in Department of Information Technology, Gudlavalleru Engineering College, Gudlavalleru, AndhraPradesh.