PORTING OF 6LOWPAN WIRELESS NETWORK STACK IN AN OPEN SOURCE OPERATING SYSTEM FOR IOT EDGE DEVICES

Ravindran.M.S[1], Geetha.D[2]
School of Electronics and Communication, REVA University, Bengaluru, India

Abstract: ***There is an explosive growth in wireless communication with sensors and actuators in homes, office buildings, factories, and even outdoors. Moreover, there is a desire to incorporate these devices as part of the Internet so that these devices could be accessed from anywhere. These devices afford new ways of communicating between each other, with —things (edge devices) outside of their own application scope, with existing infrastructure and ultimately the outside world. In a nutshell, a broad vision for the IoT is therefore for everything we might need, whether we currently know it or otherwise, to be individually accessible across the Internet.***

From this perspective, embedding a TCP/IP stack into the sensing and acting devices seems an attractive idea, which is reinforced by the new features IPv6 provides (such as the large address space and address auto configuration). Typical sensor devices are equipped with 8-bit microcontrollers, code memory on the order of 100 kilobytes, and less than 20 kilobytes of RAM. However, the TCP/IP protocol suite was not originally intended for such devices; its requirements for the underlying link layers are generally too strong to be carried out by resource-constrained devices, while certain network layer features are too complex and resource consuming.

For these reasons the IETF defined 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), [4] an adaptation layer which intermediates between the network and the link layers to provide all the services that the network layer requires but the link layer cannot provide. Wireless sensor networks are composed of large numbers of tiny networked devices that communicate untethered. For large scale networks, it is important to be able to download code into the network dynamically. It is considered that Contiki OS an open source [7], a lightweight operating system with support for dynamic loading and replacement of individual programs and services.

Contiki is built around an event-driven kernel but provides optional preemptive multithreading that can be applied to individual processes. We show that dynamic loading and unloading is feasible in a resource constrained environment, while keeping the base system lightweight and compact.

I. INTRODUCTION

Architecture diagram

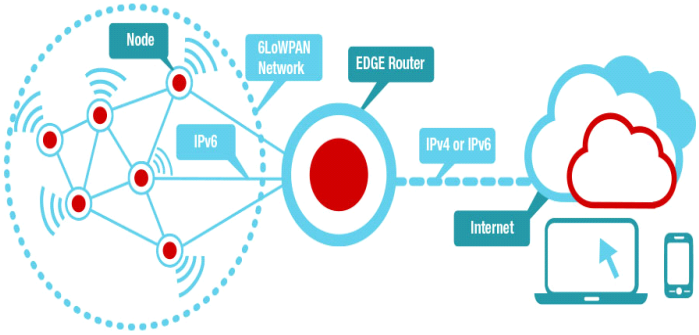

Figure 1: Architecture of IoT which includes Edge devices (nodes), Gateway (Edge router) with a connectivity to internet cloud.

As we see from the figure 1, cluster of small red nodes represent Edge devices(also called nodes). Various sensors are connected to node. Each of the nodes is been flashed with an embedded software, which includes an opensource OS containing 6LoWPAN wireless protocol stack and also the application.When there are more than one node is present in proximity, each node handshake with other and together they form a 6LoWPAN network. The big red circle represent a gateway(also called Edge router).Usually Gateway is a much powerful board consisting of an SoC and loaded with operating systems like Linux.Edge router is connected with internet which is a cloud to upstream the data from edge node through edge router. Figure 2 shows the block representation of Node, Gateway and Cloud entities.

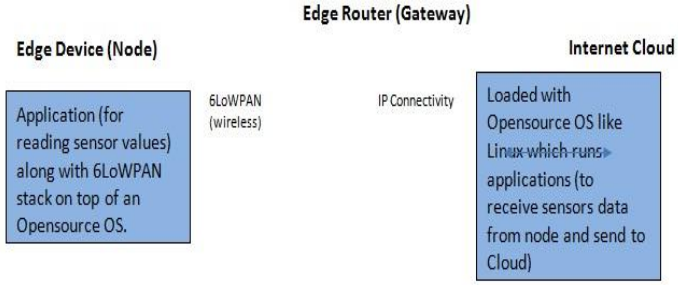

Figure 2: Schematic block representation of key entities in IoT framework.

The objective of this paper is to ultimately develop the drivers [9] for all the sensors present on the Edge node i.e. CC2650 (Things) [8] in Contiki OS [7] and then make an application which sends the value of all the sensor to our Gateway or edge router i.e. AM335x Board. This application sends the Data using 6LoWPAN. This paper consists of all the processes from the initial literature study, to the model of implementation of Application of 6LoWPAN, passing through the necessary steps of incorporating Drivers [9] for the sensors

II. RELATED WORK

It is foreseen that the future Internet is an IPv6 network interconnecting traditional computers and a large number of smart objects. This Internet of Things (IoT) will be the foundation of many services and our daily life will depend on its availability and reliable operation. Therefore, among many other issues, the challenge of implementing secure communication in the IoT must be addressed. In the traditional Internet, IPsec is the established and tested way of securing networks. It is therefore reasonable to explore the option of using IPsec as a security mechanism for the IoT. Smart objects are generally added to the Internet using IPv6 [1] over Low-power Wireless Personal Area Networks (6LoWPAN), which defines IP communication for resource-constrained networks. Thus, to provide security for the IoT based on the trusted and tested IPsec mechanism, it is necessary to define an IPsec extension of 6LoWPAN. In this paper, we present such a 6LoWPAN/IPsec extension [2] and show the viability of this approach. We describe our 6LoWPAN/IPsec implementation, which we evaluate and compare with our implementation of IEEE 802.15.4 link-layer security [6]. We also show that it is possible to reuse crypto hardware within existing IEEE 802.15.4 transceivers for 6LoWPAN/IPsec. The evaluation results show that IPsec is a feasible option for securing the IoT in terms of packet size, energy consumption, memory usage, and processing time. Furthermore, we demonstrate that in contrast to common belief, IPsec scales better than link-layer security as the data size and the number of hops grows, resulting in time and energy savings. Copyright © 2012 John Wiley & Sons, Ltd.

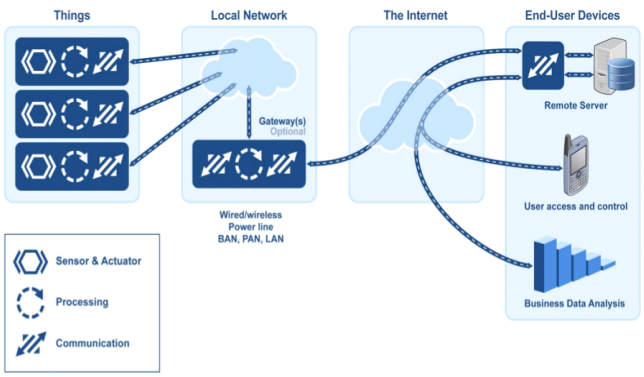

Figure 3: General setup of Internet of things

Figure 3 depicts various elements of internet of things which are in recent development relevant to contemporary IoT setup. Things are nothing but the end or edge devices where sensors are connected to. Data from things are typically sent to Local network which is a gateway. The communicateion medium from Things to Local network is usally through wireless. Local network has wired or wireless connectivity to internet. Cloud data in an internet can be accessed by various means such as the Mobile, PC, remote servers etc.

III. 6LOWPAN BAED WIRELESS NETWORK MODEL

6LOWPAN layer

6LoWPAN is an intermediate layer that allows the transport of IPv6 (see Section 2.4.2) packets over IEEE 802.15.4 (see Section 2.5) frames. Although the term 6LoWPAN [2] stands for IPv6 over Low-power Wireless Personal Area Networks. Figure 2.7 depicts how an IPv6 packet is encapsulated into a IEEE 802.15.4 frame using the 6LoWPAN adaptation layer. The IPv6 standard defines certain requirements for the link-layers over which it is to be transported. However, the IEEE 802.15.4 MAC layer does not fulfil these requirements in certain points. Hence, the 6LoWPAN specification defines not only the frame format for the transmission of IPv6 packets over IEEE 802.15.4, but also the mechanisms to obtain a unique IPv6 address from either, 16-bit or 64-bit IEEE 802.15.4 MAC addresses (using Stateless Address Auto configuration—defined in RFC 4862), and to overcome the limitations of IEEE 802.15.4. 16

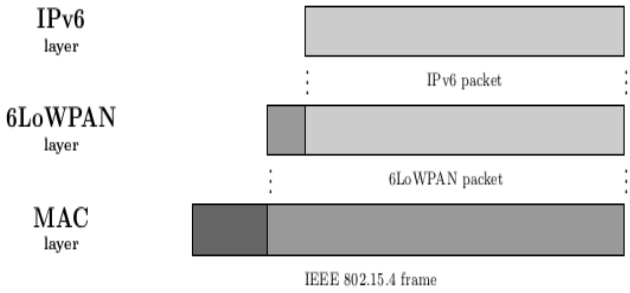

Figure 4: 6LoWPAN intermediate layer.

6LoWPAN Motivations

- Usage of IPv6 to make use of internet protocols
- Leverage on the success of open protocols in contrast to proprietary solutions
- Sensors are likely to have restricted wireless connectivity [3]
- Using IPv6 instead of something proprietary allows the usage of existing and proven protocols driving the internet
- A fully unmodified TCP/IP stack might clash with hardware limitations (which are useful for power savings)
- Sensor only need to transfer little data, compared to the usage scenarios of a Smartphone, PC.

The minimum Maximum Transfer Unit (MTU) required for a link-layer transporting IPv6 packets is, as defined in RFC 2460, 1280 octets. This is far beyond the maximum IEEE 802.15.4 frame size, which is 127 octets. Of these 127 octets, the maximum MAC header size is 25 octets and, if IEEE 802.15.4 link-layer security is enabled, it may use up to 21 additional octets. This leaves only 81 octets available for IPv6 transport. As the IPv6 header length is 40 bytes, only 41 bytes are available for transport layers and so on. In order to meet the IPv6 minimum MTU requirements, 6LoWPAN defines a fragmentation and reassembly mechanism that allows splitting IPv6 packets at the 6LoWPAN adaptation layer into smaller fragments that can be handled by the link-layer, with this process being transparent to the Internet layer.

However, applications using 6LoWPAN are not expected to use large packets, hence, in order to avoid fragmentation as much as possible, 6LoWPAN defines an IPv6 header compression mechanism .

IV. MOTIVATION FOR PORTING AN OPERATING SYSTEM(OS) TO AN IOT EDGE DEVICES

IoT consist of a huge number of single nodes (at edge). They can be used for a wide range of applications. For all different application areas different requirements have to be fulfilled. While in one application it may be most important that the nodes can operate unattended for very long periods of time, in another application they may have to be able to process huge amounts of data in short time frames. Therefore it is of high importance to choose the right hard ware and software components for the particular application. The operating system is one of the most important parts on the software side of this decision process. There are a lot of sensor node operating systems available. The Internet of Things (IoT) embodies a wide spectrum of machines ranging from sensors powered by 8-bit microcontrollers, to devices powered by processors equivalent to those in entry-level smart phones. Neither traditional operating systems (OS) currently running on Internet hosts, nor a typical OS for sensor networks are capable to fulfill all at once the diverse requirements of this wide range of devices. Hence, in order to avoid redundant developments and maintenance costs of IoT products, a novel, unifying OS is needed. The wireless sensor nodes (often called motes) usually have a microcontroller as a CPU that is not very powerful because the main focus of those motes lies in minimal power consumption since they are often designed to run on battery power for very long periods of time. And even though the microcontroller and all other components of motes are designed as low power devices, running them all at full power at all times would still consume way too much energy. So for that matter the main focus of those operating systems is energy conservation optimal usage of limited resource.

V. OPEN SOURCE OS MODEL

Comparison of different OS for IoT

In open source OS worlds,especially for IoT, the two dominant OS are Contiki [7] and TinyOS. Both provide implementations of various algorithms, protocols, device drivers, and helpful tools such as file systems or a shell. On more traditional devices connected to the Internet, the most widespread OS are Windows, several UNIX derivatives, and Linux. TinyOS and Linux are implemented as a monolithic kernel, while Contiki is built in a modular way that corresponds to a layered system. In TinyOS a set of required components is glued to together to build a single, static binary. The components expose one or more interfaces and communicate via commands and events. While the Linux kernel itself is monolithic, it is possible to configure device drivers as modules. In this way, a Linux system can be trimmed down to match exactly the particular needs for the application. But despite the fact that these modules can be loaded and unloaded during runtime, a failing driver might still crash the whole system. Contiki offers the OS facilities, such as device drivers, communication, and sensor data handling as services. Besides the mandatory components, the Contiki core however comprises also the uIP stack, a device driver loader, and the protothreading system. The scheduling in Contiki is purely event driven, similar to that in TinyOS where a FIFO strategy is used. Their scheduling strategies are optimized for simple event processing, such as handling interrupts from an asynchronous sensor. Linux currently uses the Completely Fair Scheduler (CFS) that guarantees a fair distribution of processing time based on a red- black-tree.

The programming models in Contiki and TinyOS are based on the event driven model, in a way that all tasks are executed within the same context, although they offer some kind of multi-threading support. Contiki provide protothreads as a light-weight and stackless implementation of simple multi-threading. Since events run to completion, no process synchronization between protothreads is possible.

Contiki uses a subset of the C programming language, where some keywords cannot be used. TinyOS is written in a C dialect called nesC. Linux, on the other hand, supports real multi- threading, is written in C and offers support for a wide range of programming and Key characteristics of Contiki, TinyOS, and Linux, ( ) Full 25 Support, (-) Partial Support, (X) No Support. The table compares the OS in minimum memory requirements for a basic application, support for programming languages, multi-threading, MCUs without Memory Management Unit, Modularity, and real- time behavior. Scripting languages.

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	-	X	-	✓	-	-
Tiny OS	<1kB	<4kB	X	X	-	✓	X	X
Linux	~1MB	~1MB	✓	✓	✓	-	-	-

Table 1: Comparison of different OS considered for IoT edge node

Choice of Contiki Open source OS

Contiki (Kon-Tiki) is an IPv6 ready, open source WSN lightweight operative system, design to be highly portable and memory efficient. Contiki is written in C programming language and has an event-driven kernel, but is also capable of handling per-process multithreading and interprocess communication, achieved by combining the benefits of both event-driven systems and pre-emptible threads. Contiki contains two communication stacks: uIP and Rime. uIP is a small RFC-compliant TCP/IP stack that makes it possible for Contiki to communicate over the Internet. Rime is a lightweight communication stack designed for low-power radios.

VI. RESULTS

The main goal as stated in objectives is to develop the driver for all the sensors present on the edge node and develop an application to collect the data from sensor of edge node and send it to gateway. So the drivers of some sensors are developed successfully and to see the results one demo application is develop which is sending the edge node data to the gateway. After the compilation of all the developed drivers and application, it will build the binary Image of Contiki. Then, porting that binary Image to the edge node can be done using the Open OCD debugger. After the porting of the Contiki OS on the edge node, it would start booting and collecting the data by using sensors and start

advertising the data to the air. Image of the advertising node. The results of some sensors data using the android application whose screenshots are shown below:

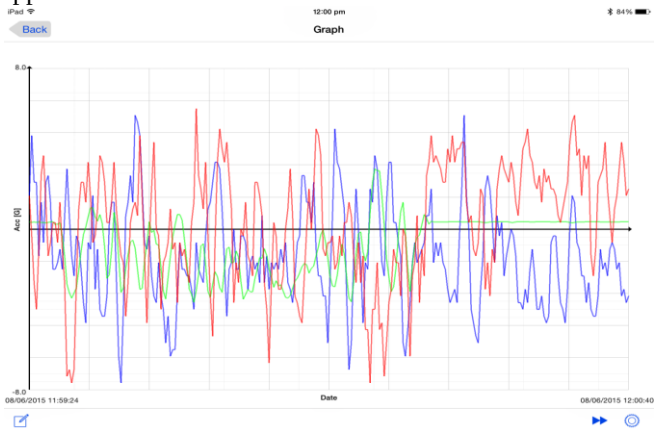

Figure 5: A typical Screenshot of the data send by accelerometer using Android Application

Accelerometer captures the movement of body in x,y and z co-ordinates. As we observe Green, Red, Blue graph indicates values of x,y, z axes as positions read through accelerometer.

VII. CONCLUSIONS

The Driver of the sensors are implemented successfully on Contiki and the data of the sensors is transmitted successfully to the gateway using 6LoWPAN and UDP protocols. [5] Contiki is also proved easy to work with, although it did see a lot of development during the course of this project, which made it slightly more difficult to test. Still, it is noted that Contiki is the Operating system offering support for TCP and C language programming. 6LoWPAN Protocol is used for transferring node data to gateway, CC2650 [8] also supports Bluetooth so that data can be transferred using Bluetooth (BLE stick) to gateway. A python scriptcan be used to see the results of the received data at the gateway node. Gateway sends the data to the cloud and that data can be accessed using web page, android application etc. In this work, the environment can be monitored using sensors at the node but in future, the environment can also be controlled using the same Operating system Contiki.

REFERENCES

[1] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, Internet Engineering Task Force, February 2006.
[2] "IPv6 over Low power WPAN (6lowpan)". IETF. Retrieved 10 May 2016.
[3] Pallàs-Areny R., "Section 2 Sensing methods and sensors", Electronic Instrumentation course, (2013).
[4] In 6LoWPAN: The Embedded Internet (Wiley, 2009), Shelby and Bormann redefine the 6LoWPAN acronym as "IPv6 over lowpower wireless area networks," arguing that "Personal" is no longer relevant to the technology.
[5] Postel, J. (1980). User Datagram Protocol. http://tools.ietf.org/html/rfc0768.
[6] IEEE Standard for Information Technology-Telecommunications and Information Exchange between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 3
[7] A Trek Through the Contiki Operating System created by Hossam Ashtawy, Troy Brown, Xiaojun Wang, Yuan Zhang Department of Computer Science and Engineering Michigan State University
[8] Data sheet of TI CC2650
[9] https://en.wikipedia.org/wiki/Device_driver

