Design and Implementation of a Smart Home Router based on Intel Galileo Gen 2

Watipatsa W. Nsunza and Xiaojun Hei

Huazhong University of Science and Technology, Wuhan, China, 430074
nsunza@hust.edu.cn,heixj@hust.edu.cn

Abstract. The emerging software defined networking (SDN) has great potential for enabling novel networking solutions to improve performance and management of distributed systems such as smart homes. In this paper, we studied the possible technical development trend of software defined wireless networking (SDWN) technologies toward the design and implementation of a smart home router based on the Intel Galileo Gen 2 programmable platform. We instrumented this platform by integrating various open-source software projects such as OpenWrt into a home router to support intelligent home wireless networking and provide low-cost connectivity solutions for the Internet-of-Things using WiFi and another cutting-edge wireless communication technology, "Bluetooth Low Energy. We conducted a series of experiments on our router and presented some preliminary results of our smart home network testbed. Our experiment study may provide empirical experiences into constructing evolvable and cost-effective software defined smart home routers with a good trade-off between performance, flexibility and cost.

Key words: Smart Home Router, Intel Galileo Gen 2, OpenWrt, Software Defined Networking

1 Introduction

In recent years, a rising number of home appliances have been equipped with communication, networking and control capabilities to provide more functional and convenient living environments [13]. Diverse user applications run on these smart devices and often require network accessibility with a home router to share a single broadband Internet access link. These home routers should be programmable in a cost-effective way to enable future evolvable smart home applications by providing quality-of-service and management interfaces to upper-layer applications.

Connectivity solutions are highly anticipated to virtualize home systems for on-demand access and easy management in a cost-effective manner to enable future evolvable smart home applications by providing quality-of-service and management interfaces to upper-layer applications. This is partly due to the Internet of Things (IoT), a massive network of all devices. The number of connected devices has been scaled at over 9 billion in 2017 and is estimated to

TRIDENTCOM 2017, September 28-29, Dalian, People's Republic of China

DOI 10.4108/eai.28-9-2017.2273386

exceed above 24 billion in the year 2020. The Housing Learning & Improvement Network "Housing LIN" published "SMART HOME - A DEFINITION" in September 2003 introduced by Intertek in their project DTI Smart Homes [5]. Intertek defines a smart home as "a dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed". The term remotely in this context refers to control both within and outside the smart home environment. Intertek lists three key elements that make a home "smart": 1) a smart home needs an internal network of wire, cable, or wireless devices; 2) it needs intelligent control from a gateway which manages the systems; 3) a smart home also needs home automation to link to services and systems outside the home.

The diversity of systems in smart homes is drastically increasing. From home appliances such as; refrigerators, microwaves, coffee makers, air conditioners and so on, to exercising equipment, smart watches, alarm clocks, security systems, lights and many other devices. All kinds of traditional devices are becoming smart and are designed to access the Internet and become part of the Internet-of-Things (IoT) [12]. The IoT is a rapidly developing application area which drives fast evolving technologies. To extend the scope of IoT devices from traditional network devices like PCs and smart phones to all kind of things such as sensors in home appliances requires an advanced connectivity solution with intelligent management. For a wireless sensor operated by coin cell batteries to send small sizes of data to the Internet, it is critical that it satisfies a certain level of energy efficiency. Without meeting this requirement, the sensors will fail over a short period of time. Bluetooth Low Energy (BLE or Bluetooth Smart) [8], is one of the latest developments in this area for short range communication. It delivers low power communications, guaranteeing sensors operate well above two years. The future of smart homes brings all network devices, sensors, and home appliances to the Internet of Things "IoT" to be accessed or controlled from anywhere. This will enhance management, security, and provide the family's necessities at their fingertips.

The introduction of open-source hardware and software platforms has enabled rapid developments in various technologies. Open licenses and source codes support collaborative projects between independent developers and companies which has increased the design scope of hardware and software developments once impossible for a single company to maintain over a long period. Multiple developers collaborate to inspect Open systems and examine security holes and other hardware and software related issues which results in rapid technology developments. The Intel Galileo Gen 2 platform evaluated in this paper is a typical system based on open-source hardware and software technologies inspired by the Arduino project. It is designed to expand the the functionality of the Arduino Uno R3 from the Arduino shield ecosystem to the Linux environment through PC standard I/O ports. This provides a simple and most cost-effective development environment when compared to other closed platforms with similar specifications.

In this paper, we studied the design and implementation of a smart home router on Intel Galileo Gen 2, to support intelligent home wireless networking and provide connectivity solutions for IoT using WiFi and BLE. More related video clips and source codes of our project can be found for reference at [9].

The organization of this paper is as follows: In Section 2 we present our system design. Section 3 describes the testbed implementation. In Section 4, we conducted a series of experiments to evaluate the system performance, and Section 5 concludes this paper and outlines some future work.

2 Testbed Design

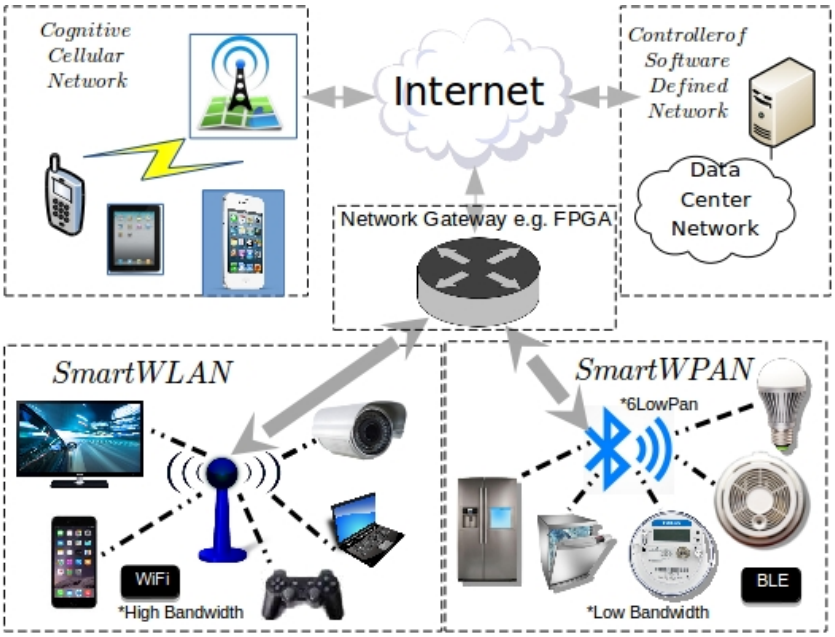


Fig. 1: A proposed software-defined edge-cloud network architecture.

2.1 Wireless Networking

We've developed our system firmware on top of "OpenWrt", a set of Open-source Linux libraries for enabling networking on embedded systems. The design scope of our smart home router incorporates the latest developments in trending wireless communication technologies. IEEE 802.11 is a set of media access control (MAC) and physical (PHY) specifications for implementing a wireless local area

network "WLAN" (also known today as WiFi). This technology operates in the 900 MHz and 2.4, 3.6, 5, and 60 GHz communication frequency bands. WiFi has become the most dominant networking technology of this age, with applications in computers, smart phones, and other bandwidth sensitive networking devices. WiFi technology is built on top of the IEEE 802.11a, 802.11b/g/n, and 802.11ac wireless communication standards. The current generation of WiFi "802.11ac" is a dual band wireless technology. It supports multiple connections at once and operates at the 2.4 and 5 GHz WiFi frequency bands. 802.11ac is also backward compatible with 802.11b/g/n wireless devices and supports data bandwidth rates up to 1300 Mbps when operating at 5 GHz and 450 Mbps at a 2.4 GHz frequency. Fig. 1 illustrates our proposed software-defined edge-cloud network architecture. In this architecture, our smart home router becomes the network gateway for both WiFi and BLE home systems.

IEEE 802.15.4 "Low Rate WPAN" (BLE) is designed for low data rate applications to efficiently manage battery consumption. This allows battery powered sensors in a Smart Home to operate for months or even years depending on their period of activity. It is a low complexity wireless standard with specifications also on both Layer 1 (PHY) and Layer 2 (MAC). BLE is highly adopted and an anticipated solution for connecting IoT devices. The restriction of the data communication topology at point-to-point, limited range of communications, and the lack of IP support make it less attractive for Internet of Things applications. The Bluetooth Special Interest Group SIG standardized the Internet Protocol Support Profile "IPSP" with IPv6 support between devices over Bluetooth Low Energy "6LowPan" [2]. The next problem arises due to the short range and restricted topology in BLE. A mesh networking protocol for multi-hop support is needed to overcome these limitations.

2.2 BLE Mesh Protocols

SIG formed a Bluetooth Smart Mesh Working Group in February 2015 which joined with Cambridge Silicon Radio "CSR" to create a global standard for Bluetooth Smart mesh[2]. The working group officially adopted profiles in 2016 and introduced "CSRmesh". This is a flood mesh protocol that utilizes the non-connectible advertisements in BLE for transmitting data to individual devices, groups, sub-groups, or all devices. CSRmesh uses the simple mechanisms of flood mesh and can communicate on a scale of up to 64,000 devices or groups per network. The flood mesh routing protocol does not need to maintain a routing table which gives CSRmesh a setup time close to zero. To support a mesh topology in BLE, CSRmesh chose a multi-layered approach built on top of BLE. This reduces the size of data payload per individual broadcasts [2, 3]. CSRmesh is not an open protocol. The specifications for this protocol show that it uses flood routing methods to connect BLE devices.

An open-source wireless mesh protocol "BLEmesh" [7] benefits from opportunistic routing. In this proof of concept, Hyun et al. proved that opportunistic flooding has higher efficiency. The experiments demonstrate that if any intermediate node receives a packet sent by the source, it is captured. The probability

of at least one node receiving the packet is 0.79, thus, approximately 1.27 broadcasts are required for the packet to reach any of the intermediate nodes. Adding the additional broadcasts performed to the packet transmission from the intermediate nodes to the destination node, 2.27 broadcasts in total are needed. Using a flood routing protocol, the source node similarly has 1.27 packet broadcasts, however the broadcasts from the three intermediate nodes total up to 1.2. The sum of the two broadcasts adds up to 2.47, so the proposed methods in BLEmesh shows that opportunistic flooding will produce optimal results.

We've designed a low energy solution to help balance network traffic by managing low bandwidth exchanges between "6LowPan" enabled devices over a separate channel. BLE support packages can be configured with newer versions of Linux above "kernel 3.17". "OpenWrt" provides the basic support packages for enabling "6LowPan" [1]. Once the BLE packages are enabled on our router, the system can be optimized to dynamically switching between BLE and WiFi for low and high Bandwidth communications depending on the stability of the power supply. An added function to this feature is the given ability to provide networking in the home during a power outage and serving in regions with non-stable power sources by adjusting the bandwidth respective to the efficiency of the power supply. This will improve connectivity in Smart Homes. Our router implementation is still in the early stages and BLE features have not yet been realized on our Intel Galileo Gen 2 platform. It has been included in this research document to serve as a reference for our future implementations.

2.3 Software Defined Networking

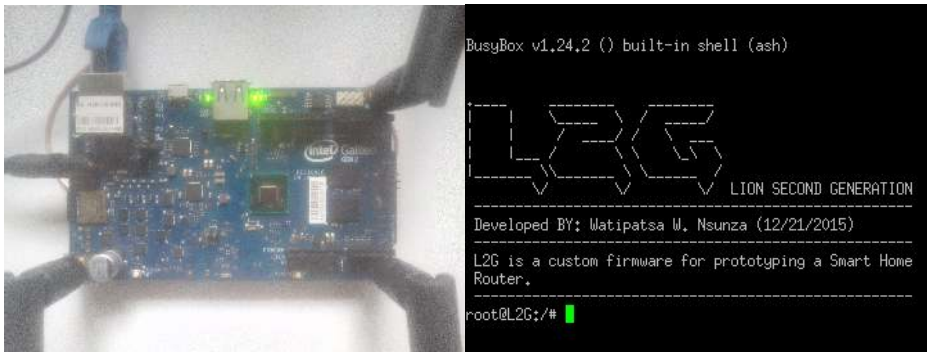
To efficiently manage network traffic from an extensible number of connected IoT devices, we integrated "OpenFlow" into our smart home router. OpenFlow is an Open-source Software Defined Networking (SDN) protocol that controls the data forwarding layers of routing [10]. SDN is a systematic shift in the networking architecture where the network control forwarding rule is disabled leaving the data forwarding layers fully programmable. This change in control provides opportunities for applications at the upper layers to seize control of the underlying network allowing them to treat the network as a logical or virtual entity enabling programmability [10].

OpenFlow is the first influential implementation of SDN [11]. OpenFlow is embedded on a central gateway such as a smart home router which downloads networking control instructions from an SDN controller to manage the network. This central unit operates as the brain of the entire network and takes charge of all packet forwarding decisions determined on a per-flow basis. All network devices are virtualized in OpenFlow which allows the OpenFlow controller to easily manage and configure each network device under a unified protocol.

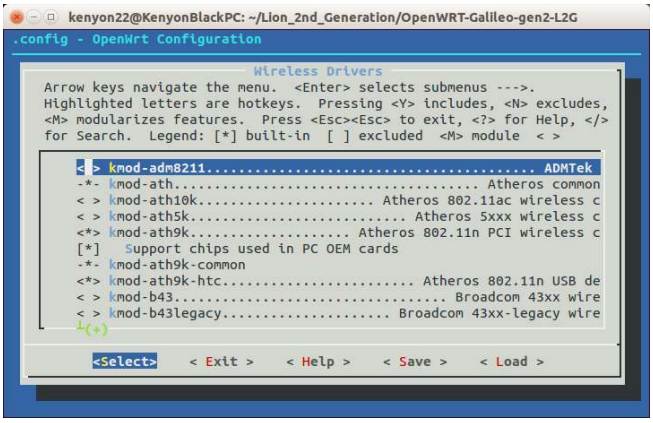
Quality-of-Service (QoS) network control scripts can be extended onto an SDN controller and improve the multimedia delivery of an embedded OpenFlow Controller. QoS improves the overall quality of network performance, in terms of error rates, bandwidth, throughput, transmission delay, availability, jitter and other performance issues. QoS has been interlinked with network virtualization.

With this elaborated testbed design, we have made our efforts to develop an open source platform for analyzing the performance of new SDN protocols to interconnect multiple communication technologies and other open source SDN projects for optimizing network performance and security.

3 Testbed Implementation



a: Intel Galileo Gen2 b: Galileo Linux Firmware



c: Firmware Development

Fig. 2: Testbed firmware development

Intel Galileo Gen 2 (Fig. 2a) is a development board based on the Intel QuarkSoC X1000 application processor with a 32-bit architecture. We developed a firmware (Fig. 2b) for this system based the OpenWrt "Trunk" source code with "Linux kernel 3.8.13, a minor update of kernel 3.8.7 based on the uClibc library. To build the OpenWrt packages, the initial step is to install OpenWrt dependencies on the build system. The build system for this project was a Linux

Ubuntu 14.04 LTS distribution running on an Intel Core2 Duo HP Pavilion Dv6-1000 computer. All packages were installed using command lines in the Linux Shell Terminal.

Once the initial packages for building an OpenWrt Image are installed, the OpenWrt configuration menu can be accessed from the Linux Shell as shown in Fig. 2c. The configuration menu presents a variety of Linux source packages that can be built into the system firmware including Ethernet, WiFi and BLE support drivers. Instructions for selecting packages are made available in this menu. We've also configured custom Linux packages into the firmware image including OpenFlow. A firmware build requires at least 10GB of free disk space. We loaded the firmware for Galileo with support for Ethernet, PCI, and USB to enable the Network and Wireless interfaces on our device.

3.1 Network Support

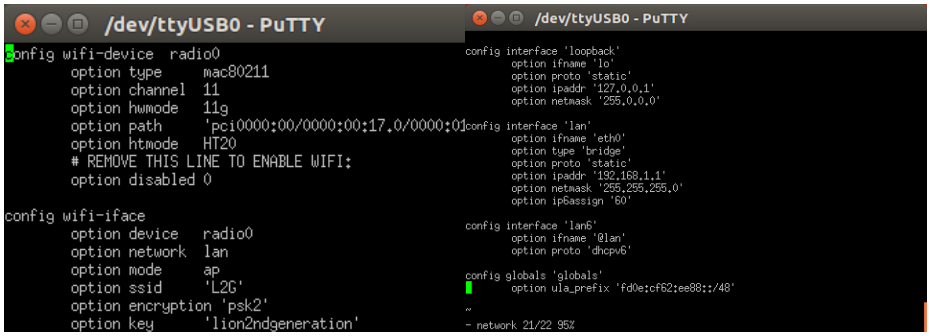
Network interfacing can be implemented on top of OpenWrt to provide Internet access by installing the required packages, driver, and configuration for the interface. OpenWrt provide instructions on enabling peripheral devices and other support packages for specific router architectures. OpenWrt and Linux developers have both contributed to support a number of peripheral devices. Since Intel Galileo is not officially supported by OpenWrt, by identifying the the peripheral interfaces on the board, we successfully installed the required drivers for the Ethernet, Wireless, and USB devices. Supported module drivers are available from the configuration menu.

Networking can be enabled for both WiFi and Ethernet by logging in the router via the Shell terminal in the firmware. Intel Galileo is a switch-less device since it only has one Ethernet interface "eth0". To enable networking, we create a "lan" interface under the Network configuration file of the firmware "/etc/config/network" to bridge all Ethernet interfaces together and set network addressing protocols as shown in Fig. 3b. We also enabled IPv6 addressing on the host by configuring "dhcp6c" not to request prefix delegation which prevents the AP from rejecting basic IPv6 addresses.

3.2 Wireless Support

We built the firmware and configured a wireless access point (WAP) based on an Atheros AR9380 PCI Card (supported by the "kmod-ath9k" driver) and the Netgear N150 WiFi Adapter (supported by the "kmod-rtl8192cu" Realtek chipset driver) available under the OpenWrt Wireless Drivers. Next, we configured the Wireless interface. Similarly after sourcing the Wireless Configuration file "/etc/config/wireless", we set some parameters for the WiFi AP (Fig. 3a). The configuration file requires values for the WiFi-device and wifi-iface on each AP. The AR9380 PCI Card interfaced on our router has 3 separate antenna channels and can support multiple APs. For this project we've only evaluated the configuration of one access point. We set the configuration of the wifi-device

under “radio0” with properties for the PCI card such as the “path” to connected bus line, number of operational “channels”, network bandwidth “modes”, etc. Using the parameters set for radio0, we set the device option for the wifi-iface and routed the option network of our WiFi AP to the lan Network interface. We set the interface mode as a WiFi AP and set the parameters for the WiFi network ID and authentication. This creates a “wlan0” wireless interface. The firewall also needs to be disabled to avoid blocking external devices from accessing the connection, the “psk2” encryption set is enough security for this network.



a: WAP Configuration b: Network Interface

c: OpenFlow inband d: OpenFlow out of band

Fig. 3: Module configuration

3.3 BLE Support

In order to add BLE support for Intel Galileo Gen 2, we’re currently upgrading the Linux kernel on Galileo from v3.8.7 to the latest official release v3.18 “Chaos Calmer”. This presents a challenge due to limited resources as we’re managing a few other projects and the current Intel QuarkSoC X1000 board support packages are only running on Kernel v3.14. The detailed steps for upgrading the Galileo Kernel will not be included in this research document. The following steps are theoretical and are for information purposes only to guide in future implementations of Bluetooth Low Energy (BLE) on Intel Galileo Gen2 and other platforms. The BLE implementation procedures in this section are

supported by another project we've completed recently and a few implementations in the OpenWrt community based on the "ASUS RT-N16" router and an "x86_64 Virtual Box" [1].

Linux Kernel 3.18 and above supports the following steps for enabling Bluetooth Low Energy "6LowPan" support on the Intel Galileo Smart Home router. This depends on a Bluetooth Support tool "bluez" consisting of the "bluez-utils" Bluetooth utilities and "bluez-libs" Bluetooth library packages. Other dependencies include "kmod-6lowpan", "kmod-bluetooth", "kmod-bluetooth_6lowpan" and USB support drivers; "kmod-usb-core, "kmod-usb-ohci", and "kmod-usb2". Once all support packages are installed onto the firmware, the bluez tool enables establishing Bluetooth Smart connections using an IPv6 stack in the software. The Intel Galileo router currently has been interfaced with a Bluetooth Smart dongle "CSR 4.0" (by Cambridge Silicon Radio) through a USB connection.

The above modules enable probing of the CSR dongle. We can then set the the PSM (Protocol/service multiplexer) channel to detect the CSR HCI device. This creates an HCI interface "hci0" for receiving advertisements from nearby 6LowPan enabled devices. The discovered BLE devices are listed in the Shell terminal using the "hcitool lescan" command. To connect to a specific device on the discovery list we echo a "connect" command containing the device IPv6 address. This requires IPv6 address support on both the router and the BLE device. The "hcitool con" command allows us to verify the connections established.

To test the IPv6 protocol on the connected device, the ping command is used to access the IP address of the device. To obtain the IP address requires a simple conversion of the BLE device MAC address. The IPv6 address of MAC "00:DD:F6:FF:20:DD" is calculated using the link-local prefix "fe80" which gives the IPv6 address "fe80::2DD:F6ff:feFF:20DD". The "ping6" command is used to ping the BLE device followed by the "%bt0" value to reference the bt0 interface for link-local addresses. Ensure that the btx device appears using the "hcitool con". Also verify whether the L2CAP CoC connection is successfully initiated and an IPv6 link to the bt"X" (X=variable) has been created to the connection interface using the "ifconfig bt0" command.

The connection procedures for BLE on the Smart Home router can be configured to automatically connect to devices in a Smart Home on control basis of the Smart Home router or manually from other control devices.

3.4 OpenFlow Setup

The following are the procedures for implementing OpenFlow v1.3 on the Intel Galileo Gen 2. OpenFlow is an external Linux package. To port this package into OpenWrt we created symbolic links of the files and packages to the root directory and the packages directory of the OpenWrt source code; see the official website of this project [9] for more detailed steps.

There are three configuration files for OpenFlow, the network interface configuration "eth0" in Section 3.1 and the wireless interface configuration file "wlan0" in section 3.2. On most OpenWrt configurations, wireless support is disabled by

default and can be enabled through command line or by setting the "option disabled" bit under wlan0 configuration file to "0". The OpenFlow configuration file is placed under "/etc/config/openflow". The OpenFlow network protocol can be configured manually to link an SDN Controller for auto-synchronization of flow entries with new classifications. Our SDN controller is still under development so we will not demonstrate the function and effect of an SDN controller in this research document. We will however describe the parameters for connecting an OpenFlow enabled router to an SDN controller. More details are available on the project website.

Fig. 3d shows the OpenFlow outband control switch configuration. The option "dp" in this configuration defines the flow data path, and option "ofports" are the available ports on the OpenFlow switch (or router). Option "ofctl" is the remote controller for the network which will later take the IP address of our SDN controller. The "mode" value changes between "inband" or "outofband", this refers to the exchange control between the controller and the forwarding devices. "Out-of-Band" control links to external networks using separate Ethernet ports for linking forward devices to the controller and for exchanging control traffic, while "In-Band" controllers use the same links for both data and control traffic.

Fig. 3c shows the inband control configuration for OpenFlow. In inband control, option "ipaddr" is the IP address configuration for the OpenFlow router, "netmask" is also the netmask configuration for the router. The option "gateway" is the IP address of the forwarding device for the network. Our Smart Home router performs both the control and forwarding role using In-Band mode. A virtual interface can also be added for inband control named "tap0" with a specified IP address and gateway route, the PC also needs to be configured with the subnet mask in order to reach this virtual interface through any OpenFlow port. This configuration can be tested by connecting the PC to any of the OpenFlow enabled ports, after configuring the Ethernet interface we can "ping" and/or telnet to the specified interface. Inband control doesn't allow routing dns and dhcp requests/replies from or to the virtual local-port (tap0), it instead requires statically setting the ipaddr option of the router using an IP address instead of URL for the controller.

3.5 QoS Setup

The QoS package can be installed on the Galileo Smart Home router using two methods. It can be installed by selecting the package in the OpenWrt configuration, or simply by installing it through command line in the OpenWrt Shell terminal on Galileo. We describe the steps for the second method in this research paper.

The "qos-scripts" package can be installed with the "opkg install" command in the shell terminal. Once the package is installed, the QoS configuration can be found under the "/etc/config/qos" directory in the firmware. After configuring the file, we can issue the script to improve the network performance using the "start" and "enable" commands in the "/etc/init.d/qos" module.

3.6 summary

Due to the present issue in upgrading the Kernel of our router, we've also not performed any OpenFlow and QoS performance tests on our Intel Galileo Gen 2 router. A full performance test will be evaluated on this router once our work is completed. OpenFlow and QoS tests on other successful architectures however demonstrate that good results can be achieved on the router.

4 Performance Evaluation

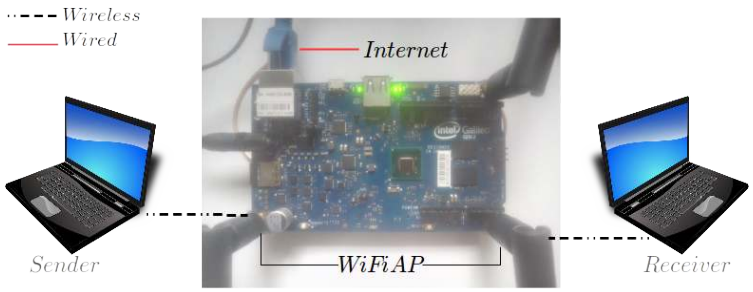


Fig. 4: Galileo Benchmark Testing Topology

In this section, we evaluate the performance on the WiFi implementation on our Smart Home router. The results demonstrate the minimum, average, and maximum throughput measurement of TCP and UDP traffic based on a ratio of packets per unit time measured in bit/sec (bps) during data transfers from sender to receiver on a single WiFi AP. Fig. 4 shows our "benchmark" evaluation setup. A benchmark testing topology consists of one router connected to the Internet and a singular access point. To setup the evaluation, we use the "Iperf " network bandwidth measurement tool on our "Linux Ubuntu" system. Two systems running Iperf were connected via a single access point and the performance is analyzed based on transmission rates over the wireless access point. The "LAN" interface on Intel Galileo was wired to a "50 Mbps" Internet connection via Ethernet and the systems were both linked to the WiFi access point, one as a server and the other as the client on Iperf. The data was analyzed on a local network from the client "receiver" end, at distinct 50 second testing phases each with approximately 100MB of data.

On a TCP connection, throughput can be affected by the upper and lower limits of the system windows, in-charge of controlling the amount of data passing through a network at a given instance. The Window size is determined by the Operating System (OS) of a device. Similarly the throughput on a UDP connection can be affected by the Buffer size of the system. Iperf can tune TCP and UDP connections by increasing the packet buffer or window size [6]. Smaller

sized windows provide poor performance on a TCP connection, but a smaller buffer size provides optimal performance on UDP. The idea is to adjust the Buffer and Window size to directly improve network performance on a physical level. Fig. 5 illustrates the TCP performance of our router for different window sizes, performed at distinct 50 second testing phases.

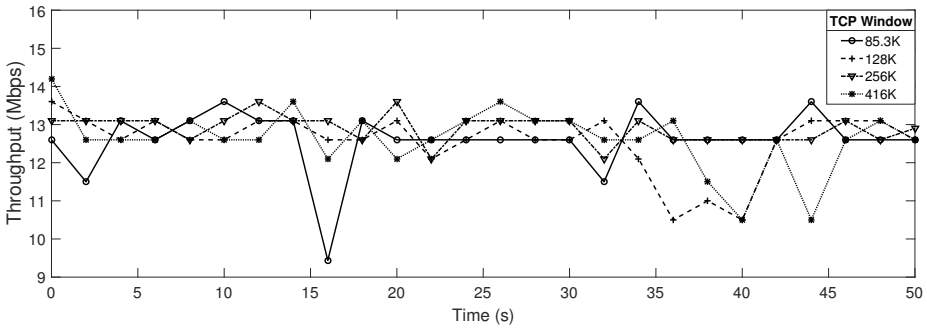


Fig. 5: TCP performance

TCP is not reliable for real-time applications, UDP is more efficient for applications such as Voice over IP (i.e. Skype), which don't require precision, but timely deliveries instead. This means that the throughput on a UDP transmission is naturally faster than that on a TCP connection. Since UDP is designed for real-time applications, a major problem arises due to packet or datagram loss. A single missing packet means the entire datagram is declared void and requires retransmission which affects the entire performance. It is much safer to transfer smaller sized packets to a larger buffer. To evaluate the Router performance on a UDP connection, using Iperf, we tune the buffer and datagram (packet) size simultaneously as shown in Fig. 6 and Table 1.

4.1 Data Analysis

Table 1 provides a summary of our routers TCP and UDP performance. The default UDP packet (annotated "D") was set at "1470" bytes and then tuned to "32768" bytes to evaluate the performance of small and large UDP buffers. The default TCP Window is set at "85.3K" while the default UDP buffer is set at "208K". The results demonstrate that small packets do not efficiently utilize a larger UDP buffer and large packets also overwhelm a small buffer, which increases Jitter levels and decreases performance due to packet loss. The best UDP performance was achieved at "15.9Mbps" on a "416K" buffer with a "32K" packet size, which resulted in increased Jitter but no packet losses. The TCP performance was however non-stable since the protocol is designed to assure delivery at a cost to the performance. The average TCP throughput is measured at "12.6Mbps".

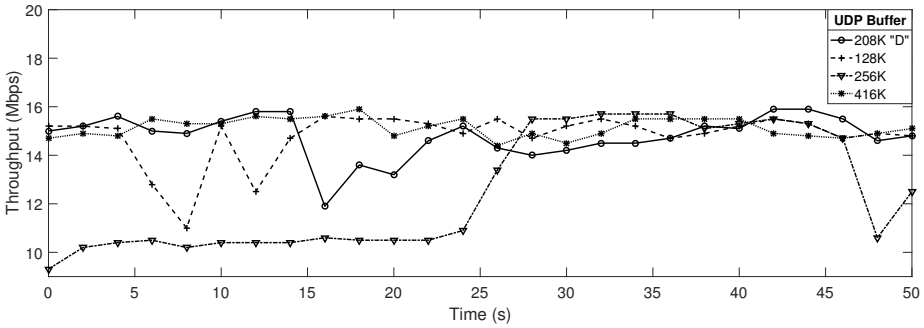


Fig. 6: UDP performance

Table 1: PERFORMANCE SUMMARY

Window/Buffer	TCP Max (Mbps)	UDP Max (Mbps)	UDP Jitter (avg.) (ms)	Packet loss % (lost/total)
85.3K	13.6	N/A	N/A	N/A
128K	13.6	15.6	22.73	85/2879 (3%)
208K "D"	N/A	15.9	5.87	1514/65362 (2.3%)
256K	13.6	15.7	27.01	1/ 2863 (0.035%)
416K	14.2	15.9	27.64	0/ 2951 (0%)

4.2 Summary

The results show that the software switch in "OpenWrt" needs much optimization in order to achieve optimal performance for both TCP and UDP. The system experienced minimal packet losses though the overall performance fluctuated during transmissions on various specifications. Performance Tests for the Intel Galileo Gen 2 Smart Home router are still being analyzed as the system continues developing. The above tests demonstrate the most basic analysis. Due to the current progress, much testing was not done on the proposed router. As research and development on this project progresses, more tests will be taken to analyze the router functionality.

5 Conclusion

In this paper, we design and implement of a smart home router based on the Intel Galileo Gen 2 programmable platform. All the proposed subjects have been addressed with as much detail as possible for information purposes to encourage more research on modern technologies and to support the development of smart homes and technology for the Internet of Things. More comprehensive evaluation experiments remain to be accomplished on this smart home testbed. In the

future, we plan to extend our study to analyze the performance of our router implementation in a smart home scenario on both "WiFi" and "BLE" interfaces using an optimized SDN protocol. In addition, it is also very important to examine the performance optimization issues for high-density WiFi networks when smart home routers are co-located in the proximity [4].

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (No. 61370231), and in part by the Fundamental Research Funds for the Central Universities (No. HUST:2016YXMS303).

References

1. IPv6 over Bluetooth smart (low energy). `https://wiki.openwrt.org/doc/howto/bluetooth.6lowpan`. Accessed: 2016-05-30.
2. Smart mesh for the smart home. `http://cwbackoffice.co.uk/docs/Rick\%20Walker.pdf`. Accessed: 2016-05-30.
3. CSRmesh wiki. `http://wiki.csr.com/wiki/CSRmesh`. Accessed: 2016-05-30.
4. Y. Gao, L. Dai, and X. Hei. Throughput optimization of multi-BSS IEEE 802.11 networks with universal frequency reuse. *IEEE Transactions on Communications*, 65(8):3399–3414, Aug 2017.
5. Smart home a definition. `http://www.housingcare.org/downloads/kbase/2545.pdf`. Accessed: 2016-05-30.
6. iPerf - the network bandwidth measurement tool. `https://iperf.fr/iperf-doc.php`. Accessed: 2016-05-30.
7. H. S. Kim, J. Lee, and J. W. Jang. BLEmesh: A wireless mesh network protocol for bluetooth low energy devices. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 558–563, Aug 2015.
8. J. Nieminen, C. Gomez, M. Isomaki, T. Savolainen, B. Patil, Z. Shelby, M. Xi, and J. Oller. Networking solutions for connecting bluetooth low energy enabled machines to the Internet of things. *IEEE Network*, 28(6):83–90, Nov 2014.
9. Watipatsa W. Nsunza. Design and implementation of a smart home router. `http://itec.hust.edu.cn/~kenyon22/FYP_2016.html`.
10. Open networking foundation (ONF), software defined networking: the new norm for networks. `https://www.opennetworking.org/images/stories/downloads/openflow/wpsdn-newnorm.pdf`. Accessed: 2016-05-30.
11. N. McKeown T. Anderson H. Balakrishnan G. Parulkar L. Peterson J. Rexford S. Shenker and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 8(2):69–74, 2008.
12. Smart homes: Past, present, and future. `http://www.wiredupinstallation.com/smart-home-business-technology/smart-homes-past-present-and-future/`. Accessed: 2016-05-30.
13. Tausif Zahid, Fouad Yousuf Dar, Xiaojun Hei, and Wenqing Cheng. An empirical study of the design space of smart home routers. In *Proceedings of the 14th International Conference on Inclusive Smart Cities and Digital Health (ICOST)*, pages 109–120, 2016.