SECURE SWITCH DEVELOPMENT USING OPEN NETWORK LINUX ON BARE METAL SWITCH, DEVELOPMENT OF NETWORK OPERATING SYSTEM FOR BARE METAL SWITCHES(NOS)

Muhammad Bilal Tajammal, Dr. Hanif Durad and Rao Nazar Iqbal
Department of Computer & Information Sciences
Pakistan Institute of Engineering & Applied Science (PIEAS)
Islamabad, Pakistan
mbilaltajammal@gmail.com, hanif@pieas.edu.pk, raonazariqbal@hotmail.com

Abstract—Information technology field is rapidly growing. No one wants their information being eavesdropped by others. This paper introduces the techniques to fulfill those needs by developing a generic network operating system based on Open Network Linux(ONL) that can be used on supported bare metal switches available in market as well as custom developed hardware switches. The developed operating system consists of code to implement layer 2 switching and layer 3 routing. This provides with not only secure code but also root (operating system) level access so user can manage and monitor every bit of information in a way no other existing devices provide. Also layer 3 switch supports both switching and routing based on both MAC and IP addresses so has less latency as compared to routers and more features as compared to switches. In our work, we used Quanta Mesh® T1048 LB9 bare metal switch for implementation purpose.

Keywords—Bare metal switch;Open Network Linux; Secure NOS;layer forwarding; Secure switch/route; Vendor Independent Switches

1. Introduction. Today the world is rapidly growing in Information technology field. Information plays a very vital role. No one wants that their information is being monitored by any third party. Secure and reliable communication is becoming need of this vastly growing IT world. This paper aims to target those needs by providing an native Layer 3 switch which will not only be secured ,as code will be written by the user itself rather than by a specific vendor and will also be reliable by deploying and using advanced networking (L2/3) protocols. Such equipments are needed by every working sector, starting from public, private, corporate sector to military institution where a secure and reliable network is required.

The communication in any computer network are governed either by layer 2 or/and layer 3. So we can make sure that at least our routers and switches don’t have any kind of backdoors in it and this is only possible if we develop customizable NOS (network operating system) of a switch so we have root level access.

Conventional switch devices operate at layer 2 of OSI model, where packets are sent to a specific switch port based on destination MAC addresses. Routing (Devices) operates at layer 3, where packets are sent to a specific next-hop IP address, based on destination IP address. Every network device uses an operating system that controls everything from packet delivery to configuration management.

Open Network Linux (ONL) is a Linux distribution used as an operating system for “bare metal” switches. It is a part of the Open Compute Project. ONL uses Open Network Install Framework (ONIE) to install onto on-board flash memory, which mostly comes pre-installed with bare metal switch. With this as install environment, different network operating systems can be installed on bare metal switches. In this paper ONL will be used as Network Operating System (NOS). Also layer 3 switches are fast as compared to traditional layer 2 switches, So this paper also aims to build a layer 3 switch that fulfils the needs by using less hardware.

Final product is a switch image file with (.swi) extension that will contain compiled ONL as NOS and then ONIE is utilized to install it on bare metal switch. The docker technology is used for compilation through prebuilt containers that has all the prerequisites to successfully compile kernel as well as other source code.

Related Work. Jiří Pírko presents the current effort to unify and uphold the Linux networking model across the spectrum of devices which is necessary to make Linux the cornerstone of industrial grade networking [1]. The scope of this paper covers state of art with current implementation of standard commodity switches such as top of rack switches, small home gateway device as well as SRIOV NIC embedded switches.
Open Network Linux community presents with the work on developing and maintaining Linux kernel specifically for networking purpose [2]. It is based on Debian kernel and has been ported to support networking devices e.g. bare metal switches from different companies. It consists of kernel and some Linux shell and also some forwarding agent based on the underlying hardware device. There are different forwarding agents like FRR, FBOOS that works on different hardware devices (Bare metal hardware). But forwarding agents depends on hardware manufacturers to give API / driver source code to port forwarding agent to that platform. Not every manufacturer gives this information.

FBOOS is developed by Facebook and currently being used on Wedge devices. It is open source and free, but not available for other platforms yet [3].

SAI (switch abstraction Interface) is an open source project that also aims to target different hardware manufacturer and encourage them to adopt a standard API mechanism to provide a vendor-independent way of controlling forwarding elements, such as a switching ASIC, an NPU or a software switch in a uniform manner [4].

There is some need for a universal forwarding agent and/or some system that can support as many ASIC chips as possible without minimum or no modification in source code.

In this paper we presented a way to overcome this situation with developing QUANTUM, system software that also works as forwarding agent as well as includes other critical parts and utilities essential running working Network operating system on bare metal Hardware. With this type of system, one has to only substitute minimum efforts to port QUANTUM system to other hardware platforms.

We used LB9 Quanta Mesh® hardware that currently does not have Layer 2 implementation and suitable forwarding agent available in open source or in ONL. So we developed and used QUANTUM as a forwarding agent.

2. Equipment & Tools used
   a. Bare Metal Switch – Quantamesh T1048-LB9

   Some salient features of the hardware device are listed in Table 2-1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switching capacity</td>
<td>176Gbps</td>
</tr>
<tr>
<td>Forwarding rate</td>
<td>131Mpps</td>
</tr>
<tr>
<td>Memory</td>
<td>1024MB DDR1</td>
</tr>
<tr>
<td>Flash</td>
<td>64MB</td>
</tr>
<tr>
<td>Packet buffer</td>
<td>4MB</td>
</tr>
</tbody>
</table>

   Table 2-1 LB9 Features

   ![Figure 2-1 Quanta mesh T1048 LB9](image)

   b. Open Network Install Framework (ONIE)

   ONIE is a small piece of software that ONL expects to exist on every switch, pre-installed by the switch vendor. ONIE provides the installation and management utilities to install/uninstall/rescue a Network Operating System (“NOS”) like ONL. While ONIE is a standalone operating system in its own right, it is intentionally stripped down and has few features outside of the bare minimum needed to bootstrap a system and invoke an NOS installer program.

   c. Open Network Linux

   Open Network Linux is a Linux distribution for "bare metal” switches, that is, network forwarding devices built from commodity components. ONL uses ONIE to install onto on-board flash memory. Open Network Linux is a part of the Open Compute Project and is a component in a growing collection of open source and commercial projects. Table 2-2 lists some of software component configurations used in this paper for implementation purpose on LB9 hardware switch.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Open network Linux</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Architecture</td>
<td>Power PC</td>
</tr>
<tr>
<td>Based on</td>
<td>Debian 7.0</td>
</tr>
<tr>
<td>Code Name Release</td>
<td>Wheezy</td>
</tr>
<tr>
<td>Kernel</td>
<td>Open Network Linux for ppc build</td>
</tr>
<tr>
<td>Kernel Version</td>
<td>3.9.6</td>
</tr>
<tr>
<td>Chipset Driver</td>
<td>Brcm-6.3.3 (API for FireBolt 3)</td>
</tr>
</tbody>
</table>

Table 2-2 Software components used

d. Open Route Cache

1. ORC maps physical routing Chip (ASIC) physical ports to Linux interfaces. That is after running, it loads the `bcm-6.3.3` module, a Broadcom driver provided by Broadcom under NDA which is a half open/half closed source system. The File `orc_brcm.so` is driver file loaded by orc daemon modules to allow access to bare metal’s physical ports and their events (port up, down).

2. It utilized the bare metal’s routing chip (ASIC chips) for routing, for this purpose when a route is added to kernel table via some high level protocol or by route command, it adds those routes to ASIC chip which makes the routing, switching very fast as it is not utilizing the CPU, but dedicated chip technology. It only implements layer 3 basic protocols. For other protocols like OSPF we have to use some other solution.

e. Quagga

Quagga is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, IS-IS, BGP-4, and BGP-4+. Only use to get protocols not natively available in kernel mode like, BGP, OSPF etc.

Quagga has an interactive user interface for each routing protocol and supports common client commands. Due to this design, you can add new protocol daemons to Quagga easily.

3. bare metal hardware. The bare metal hardware is specialized computing devices striped down to use high end processing chips designed for networking tasks. The result will be 100 of gigs per second as compared to other computers.

1. Slow Path vs. Fast path

   *Slow path* is path that uses CPU in processing. It involves OS involvement.

   *Fast path* is when data go through ASIC chip as opposed to general CPU path.

2. Lower cost in case of bulk production

3. SOC (system on chip) support i.e. ASIC (co)-processors

4. Hardened systems

5. Lower maintenance cost

In Figure 3-1 the current vendor model being used is shown in.
We want to develop a more generalized model that is by only replacing driver for a specific chip the OS can be used on any other BMS. The proposed model is show in Figure 3-2 below:

**4. Design & development**

   **a. QUANTUM**

Quantum is system software developed entirely for this paper purpose. It is responsible for implementing layer 2 (switching) and layer 3 (routing) functionality for in the system. Its code is written in C language for that
directly interacts with OS kernel and ASIC hardware so the desired functionality of ASIC and OS can be utilized and some modules are written in Lua programming language. It is also responsible for work as forwarding agent.

The Figure 4-1 shows inside architecture of QUANTUM project.

![QUANTUM Architecture](image)

The Details of Modules are given below:

i. Memory Manager

Memory manager is responsible for the allocation and de-allocation of memory. It operates in 2 modes

a. It defines a pool of memory for high throughput and continuous allocation and de-allocation. It gets large chunk of memory from OS and then allocate to program when needed.

b. It also redefines default `malloc` library function to add support for auto free when memory no longer required and also prevent pointer dangling error.

ii. Web Server

This consists of a web server module that provides web management console so end user can manage switch using GUI instead of using command line based consoles. This is useful as not every user is familiar with command syntax and other details while using command based console. Below are some figures (Figure 4-2, Figure 4-3 Figure 4-4) of Web management console.
Figure 4-2 Web Management Console Login Page

Figure 4-3 Switch status page

Figure 4-4 FAN and sensors readings
iii. Driver Management
This module is written to directly interact with switch ASIC hardware chip to utilize its functionality. This consists of two layers
1. User space driver
   This module calls the Broadcom provided kernel driver that contains functions to control functionality. It loads this driver in kernel space and then calls it using pointer notations to invoke functions.
2. Kernel space driver
   This module is kernel space driver that controls OS level switch functions e.g. universal support for establishing layer 2 bridge functionality without using any third party APIs. This can be used as a universal switching and routing system that interact directly with hardware registers and tables and will run the system without third party APIs need.

iv. Core System
The core system consists of multiple functional modules.
a. CLI
The command line interface is a program written in C that intercept the user typed command and then return with appropriate functionality. There are many commands that can be used to configure switch. Some outputs are shown in Figure 4-5, Figure 4-6 and Figure 4-7

---

![Quantum Build Details](image1)

**Figure 4-5 QUANTUM Build Details**

![Quantum Options](image2)

**Figure 4-6 QUANTUM options**
b. **Log manager**
The purpose of log manager is to write and read from log files and preserve every action using log files so it can be tracked.

c. **Configuration Manager**
The configuration manager is the module that interacts with the system and configures its parameter e.g. interfaces, IPs, Macs, databases etc. by reading from configuration files. The current configuration file is in default INI file format.

d. **Daemon subsystem**
The daemon subsystem is used by logger and configuration manager and also responsible to track system stability and report in case of crash or other problems.

5. **Implementation**. Implementation is not a difficult task as our current BMS supports ONIE. So the following steps are followed.

a. **ONIE Update**
First The Default ONIE is updated with custom modified ONIE code. Figure 5-1 shows process output.

b. **ONL install**
Our final developed ONL image with layer 2 and layer 3 managers is then installed using standard procedure of installing OS using ONIE. Figure 5-2 shows installation output.
The OS is now installed and running.

6. **Results & Discussions.** The results of our experiments are given below.

   a. **Ping Test**

   First connected the switch with a system with some IP address say 192.168.2.195 and gateway has IP address of 192.168.2.2 Plugged on cable on port 1, and from gateway plugged in port 2 and then ping the gateway. Result is shown in Figure 5-3.

   ![Figure 5-3 ping results](image)

   b. **ARP Table**

   The ARP Table on layer 2 is now at work. While connect a port to external network and then system can access internet as well as other computers on the network as a simple layer 2 implemented switch like Cisco will do. ARP table is shown in Figure 5-4.
c. Throughput test at layer 2 for 48-ports

The throughput is measured by enabling all ports and then running script that sends file from one system to another. Results are shown in. These test are taken by using slow path e.g. port > CPU > kernel > port. So results will be slow as expected.

Results are shown in Table 5-1. Graphical Representation is shown in Figure 5-5 Throughput Graph.

<table>
<thead>
<tr>
<th>Group</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Time (M)</th>
<th>Relative Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>4.05</td>
<td>0</td>
<td>0</td>
<td>00:00</td>
<td>0</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.92</td>
<td>3.81</td>
<td>4.01</td>
<td>03:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.821</td>
<td>3.81</td>
<td>4.056</td>
<td>06:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.9</td>
<td>3.81</td>
<td>4.056</td>
<td>13:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.89</td>
<td>3.81</td>
<td>4.056</td>
<td>20:00</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.91</td>
<td>3.81</td>
<td>4.056</td>
<td>26:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>3.90</td>
<td>3.81</td>
<td>4.056</td>
<td>35:00</td>
<td>0.118</td>
</tr>
</tbody>
</table>

Table 5-1 Throughput measurement data

d. Transaction Rate

Transaction time is measured by continuously ping the system. PING result is fairly good. Results are shown in Table 5-2. Graphical Representation is shown in Figure 5-6.

<table>
<thead>
<tr>
<th>Group</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Time (M)</th>
<th>Relative Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>0.05</td>
<td>0</td>
<td>0</td>
<td>00:00</td>
<td>0</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>03:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>06:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>13:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>20:00</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>26:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>0.048</td>
<td>0.048</td>
<td>0.051</td>
<td>35:00</td>
<td>0.118</td>
</tr>
</tbody>
</table>

Table 5-2 Transaction Data
e. **Response Time**

Response time is measured by starting data sent/receive at different ports and then ping other system at the same time. Results are shown in Table 5-3. Graphical Representation is shown in Figure 5-5 Throughput Graph.

<table>
<thead>
<tr>
<th>Group</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Time (M)</th>
<th>Relative Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>10</td>
<td>8.99</td>
<td>11.03</td>
<td>00:00</td>
<td>0</td>
</tr>
<tr>
<td>Pair 1</td>
<td>20.935</td>
<td>19.925</td>
<td>21.965</td>
<td>03:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>19.935</td>
<td>18.925</td>
<td>20.965</td>
<td>06:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>18.95</td>
<td>17.94</td>
<td>19.98</td>
<td>13:20</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>21.08</td>
<td>20.07</td>
<td>22.11</td>
<td>20:00</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>20.01</td>
<td>19</td>
<td>21.04</td>
<td>26:40</td>
<td>0.118</td>
</tr>
<tr>
<td>Pair 1</td>
<td>19.01</td>
<td>18</td>
<td>20.04</td>
<td>35:00</td>
<td>0.118</td>
</tr>
</tbody>
</table>

Table 5-3 Response Time data

7. **Conclusion.** ONL is a Linux distribution for bare metal switches. ONL uses ONIE to install onto board flash memory. Using ONIE we can install NOS of our own choice. To deal with the rising cyber security issues a secured switch has been made using ONL as NOS and custom developed application/patches hence both layer 2 and layer3 has been achieved. Protocols like SNMP, STP, IGMP, and ARP have been implemented successfully. Further improvements can be done by developing other application on layer 2/3. Better and more efficient protocols can also be further developed. Complete hardware can also be made by purchasing ASIC only rather than buying entire bare metal switch.

8. **Future Work.** Some future work suggestions are given below:

1. **IDS/IPS solution**

   Now that we has switch root level access so any development/configuration can be done on it as it is a normal computer with Linux installed. There are plenty of packages for PowerPC and custom can be developed. So it is possible to port snort/suricata to switch at layer 2/3 with minimal efforts.
2- **SDN solution**

The switch is ready for SDN deployment. Currently ONL provided OFDPA is not ported to LB9 platform. But as it is open source and code is available, it can be ported to ONL with some serious efforts so we can use switch as SDN ready with indigo or other means. In current situation, POX/NOX can be used to build SDN over switch ONL OS, but this is not recommended as it will result in slow performance.

3- **Universal Driver architecture**

This module that is under active development is a kernel space driver that controls low level switching functions e.g. universal support for establishing layer 2 bridge functionality without using any third party drivers. On completion this can be used as a universal switching and routing system that interact directly with hardware registers and tables and will run the system without third party API need. Of course the chip registers and table(s) information must be needed from vendor and/or other sources before using this. Hence organizations can use any hardware they desire with this switch OS without any code level modifications and with minimum loss in speed and functionality.

4- **ONL 2.0 Support**

The support for ONL version 2.0 is under development. It has some feature like actual file system, resume capability, no overlay file management, fast boot up etc, easy upgrade etc.

**References**


