Simulation Of Networking Protocols On Software Emulated Network Stack

Hrushikesh Nimkar, Bhavesh Munot, Saish Sali, Vipul Rathore

Abstract: With the increasing number and complexity of network based applications, the need to easy configuration, development and integration of network applications has taken a high precedence. Trivial activities such as configuration can be carried out efficiently if network services are software based rather than hardware based. Project aims at enabling the network engineers to easily include network functionalities into his/her configuration and define his/her own network stack without using the kernel network stack. Having thought of this, we have implemented two functionalities UPNP and MDNS. The multicast Domain Name System (MDNS) resolves host names to IP addresses within small ad-hoc networks and without having need of special DNS server and its configuration. MDNS application provides every host with functionality to register itself to the router, make a multicast DNS request and its resolution. To make adding network devices and networked programs to a network as easy as it is to plug in a piece of hardware into a PC, we make use of UPnP. The devices and programs find out about the network setup and other networked devices and programs through discovery and advertisements of services and configure themselves accordingly. UPNP application provides every host with functionality of discovering services of other hosts and serving requests on demand. To implement these applications we have used snabbswitch framework which an open source virtualized ethernet networking stack.

Keywords: Software Defined Network, Network Virtualization, Network Protocols, UPnP, mDNS.

INTRODUCTION

In this project, we are going to simulate some network protocols on software emulated network stack which is nothing but Snabbswitch. Snabbswitch is a network stack which runs completely in user space and provides the basic functionalities required for networking. Snabbswitch has its own device driver for fetching the network packet from NIC card to Snabbswitch Framework and hence it completely bypasses the Kernel which is the significant part. The motivations behind using software emulated network stack are, first, user space applications always outperform the Kernel space applications. Second thing is implementing any protocol or application is always easier in user space than in kernel space. Also packet processing is faster in user space than in kernel space. Here, in this project, we have simulated following protocols in this network stack:-

1. Universal Plug and Play (UPnP)
2. Multicast Domain Name System
3. Basic Firewall
4. Packet Content
5. Port Forwarding

SNABBSWITCH

Snabbswitch framework, as stated earlier, runs in user space and bypasses the kernel leading to improved performance. Lua is the language used to implement this framework.

Why Lua?

Snabbswitch framework, entirely, is written in Lua language. Lua is dynamical typed language with the performance as good as as system level languages like C and C++. Lua use the LuaJIT tool i.e. Lua Just-In-Time compiler. Lua has a FFI library which enables us to call C language functions too. Snabbswitch has the 4 main components in it.

1. Apps
2. Links
3. Config
4. Engine

1. App: App is basically a name given to any functionality. These apps act as nodes - which takes packet as input processes them and outputs the processed packet – in the system view. Also, these apps can be used as building block of some new functionality i.e. small basic apps can be used to develop some larger functionality. An app is the implementation of some specific networking function. For example, a switch, a router, or a packet filter etc. Apps receive packets on input ports, perform some processing, and transmit packets on output ports. Each app has zero or more input and output ports. For example, a packet filter may have one input and one output port, while a packet recorder may have only an input port.

2. Links: Links are nothing but ring buffers containing packets. Links can be treated either like arrays -- accessing their internal structure directly -- or as streams of packets via API functions. Links act as the connection between two apps.

3. Config: A config is a description of a packet-processing network. The network is a directed graph. Nodes in the graph are "apps" that each process packets in a specific way -- switch, route, filter, police, capture, etc. Each app has a set of named input and output "ports" -- for example called rx and tx. Edges in the graph are unidirectional "links" that carry packets from an output port to an input port. The config is a purely passive data structure. Creating and manipulating a config object does not immediately affect operation. The config has to be activated using ‘engine. configure(c)’.

4. Engine: The engine executes a configuration by initializing apps, creating links, and driving the flow of execution. The engine also performs profiling and reporting functions. It can be reconfigured on-the-fly.

BLOCK DIAGRAM OF FRAMEWORK

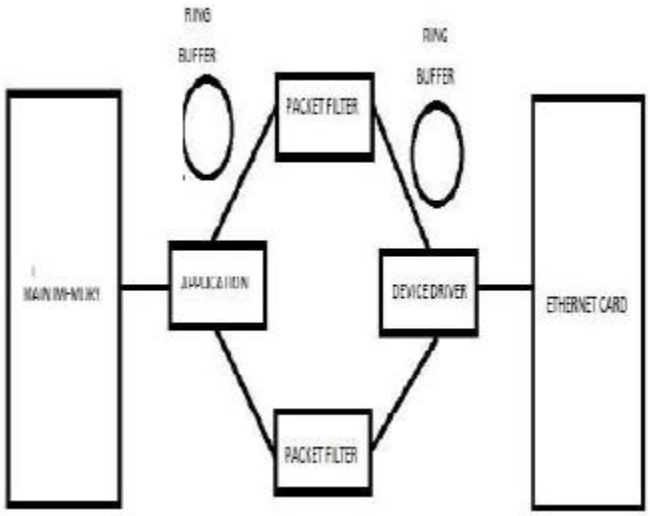

Fig.1: *Block Diagram of Snabbswitch Framework*

1. App:
'myapp:new(arg)' Create an instance of the app with a given argument. `myapp.input` and `myapp.output` Tables of named input and output links. These tables are initialized by the engine for use in processing. `myapp:pull()` Pull new work into the system. (Optional.) For example: input packets from the network and transmit them to output ports. `myapp:push()` Push existing work through the system. (Optional.) For example: move packets from input ports to output ports or onto an external network. `myapp:relink()` React to a change in input/output links. Called after a link reconfiguration and before the next packets are processed. `myapp:reconfig(arg)` Reconfigure with a new arg. (Optional.): recreation of the app is used as a fallback.) `myapp:report()` Print a report of the current status. `myapp.zone` Name of the LuaJIT profiling zone for this app (a descriptive string). (Optional: module name used as a default.)

2. Links:
`link.empty(l)` Return true if the link is empty. `link.full(l)` Return true if the link is full. `link.receive(l)` Return the next available packet (and advance the read cursor). If the link is empty then an error is signaled. `link.transmit(l, p)` Transmit a packet onto the link. If the link is full then the packet is dropped and the drop counter increased). `link.stats(l)` Return a structure holding ring statistics:`txbytes` and `rxbytes` count of transferred bytes.`txpackets` and `rxpackets` count of transferred packets. `txdrop` count of packes dropped due to ring overflow.

3. Config:
`config.new() => <config>` Create a new empty configuration. `config.app(c, name, class, arg)` Add an app to the config. Example: `config.app(c, "nic", Intel82599, {pciaddr = "0000:00:00.0"})` `config.link(c, linkspec)` Add a link from an output port to an input port. The linkspec is a string with syntax is `from_app.from_port->to_app.to_port`. Example: `config.link(c, "nic1.tx -> nic2.rx")`.

4. Engine:
`engine.configure(c)` Configure the engine to use a new configuration. `engine.main(options)` Execute the engine. `duration`: number of seconds to execute (a floating point number). 'engine.report()` Print a report on current operational state.

WHAT HAVE WE DONE?

In this project we are going to simulate the working of - UPnP, mDNS, Port Forwarding, Firewall, Packet content – these applications. The original aim was to write these applications and port them for Raspberry Pi. But the problem which we faced is that snabbswitch framework has device driver written for Intel10g card and Raspberry Pi has a Broadcom chip. Thus we reduced the scope of our project a little and simulated the working of these protocols.

Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) is a set of networking protocols that permits networked devices, such as personal computers, printers, Internet gateways, Wi-Fi access points and mobile devices to seamlessly discover each other's presence on the network and establish functional network services for data sharing, communications, and entertainment. UPnP is intended primarily for residential networks without enterprise-class devices. The concept of UPnP is an extension of plug-and-play, a technology for dynamically attaching devices directly to a computer, although UPnP is not directly related to the earlier plug-and-play technology. UPnP devices are "plug-and-play" in that when connected to a network they automatically establish working configurations with other devices.

Protocol:
UPnP uses common Internet technologies. It assumes the network must run Internet Protocol (IP) and then leverages HTTP, SOAP and XML on top of IP, in order to provide device/service description, actions, data transfer and eventing.

1. Addressing:
The foundation for UPnP networking is IP addressing. Each device must implement a DHCP client and search for a DHCP server when the device is first connected to the network. If no DHCP server is available, the device must assign itself an address.

2. Discovery:
Once a device has established an IP address, the next step in UPnP networking is discovery. The UPnP discovery protocol is known as the Simple Service Discovery Protocol (SSDP). When a device is added to the network, SSDP allows that device to advertise its services to control points on the network.

3. Description:
After a control point has discovered a device, the control point still knows very little about the device. For the control point to learn more about the device and its capabilities, or to interact with the device, the control point must retrieve the device's description from the location (URL) provided by the device in the discovery message.

4. Control:
Having retrieved a description of the device, the control point can send actions to a device's service. To do this, a control point sends a suitable control message to the control URL for

the service (provided in the device description).

Multicast Domain Name System (mDNS)
The multicast Domain Name System (mDNS) resolves host names to IP addresses within small networks that do not include a local name server. It is a zero configuration service, using essentially the same programming interfaces, packet formats and operating semantics as the unicast Domain Name System (DNS). While it is designed to be stand-alone capable, it can work in concert with unicast DNS servers.

Protocol:
When an mDNS client needs to resolve a host name, it sends an IP multicast query message that asks the host having that name to identify itself. That target machine then multicasts a message that includes its IP address. All machines in that subnet can then use that information to update their mDNS caches.

Port Forwarding
In computer networking, port forwarding or port mapping is an application of network address translation (NAT) that redirects a communication request from one address and port number combination to another while the packets are traversing a network gateway, such as a router or firewall. This technique is most commonly used to make services on a host residing on a protected or masqueraded (internal) network available to hosts on the opposite side of the gateway (external network), by remapping the destination IP address and port number of the communication to an internal host. Port forwarding allows remote computers (for example, computers on the Internet) to connect to a specific computer or service within a private local-area network (LAN).

Packet Content
Packet content app is used to display the different fields in packet structure. By parsing the content of packet by using the offsets of field – from packet structure – we display the content of packet like source IP address, destination IP address etc.

Firewall
In computing, a firewall is a network security system that controls the incoming and outgoing network traffic based on an applied rule set. A firewall establishes a barrier between a trusted, secure internal network and another network (e.g., the Internet) that is assumed not to be secure and trusted. Firewalls exist both as software to run on general purpose hardware and as a hardware appliance. Many hardware-based firewalls also offer other functionality to the internal network they protect, such as acting as a DHCP server for that network. Many personal computer operating systems include software-based firewalls to protect against threats from the public Internet. Many routers that pass data between networks contain firewall components and, conversely, many firewalls can perform basic routing functions.

MATHEMATICAL MODEL

Let S be the system such that-
S = {VNS, NIC, DD | Φ}
Where,
VNS is Virtual Network Stack.
VNS consists of different applications which are connected using links to form a stack.

VNS = {UPnP, mDNS, Firewall, Packet Content, Port Forwarding}

Firewall = {I, O, F, SUCCESS, FAILURE}
F2 = Reading packets from the input links.
F3 = Reaching to the required offsets and ten reading the relevant information from packets

SUCCESS = {Correct Information is Displayed}

FAILURE = {Required information is not present or is corrupt}

mDNS = {Host, Router}
where,
Host = {I, O, F, SUCCESS, FAILURE}
where,
I = {R1, R2…Rn, RS1, RS2...RS3}
[Host receives packets which are request from other hosts or response to its own request]

O = {R1, R2…Rn, RS1, RS2..RS3, RG1…RGn}

[Host is capable of sending requests and responses to requests and send registration packets to the router]

F = {F1, F2, F3...Fn}
[Functions of Host]
Where,
F1 = Register itself to the router. F2 = Make a request packet to get the IP address of particular host.
F3 = Respond to a request sent by other host.

SUCCESS = {Registration is successfully done, Request is sent and response is received}

FAILURE = {The Response is not received if the host name is not present}

Router = {I, O, F, SUCCESS, FAILURE}

Where,
I = {P1, P2, P3...Pn, RG1...RGn}
[Router receives packets which help the host register itself to the router and other packets are request/response which it simply forwards.]

O = {P1, P2, P3,..Pn}

Where,
I = {Rules of filtering, IP0, IP1, IP2...IPn}
[Rules for filtration and IP Packets from different sources]

O = {IP0, IP1, IP2...IPn}
[Processed IP Packets]

F = {F1, F2, F3…Fn}
[Functions of Firewall Application]
Where,
F1 = Keep Reading packets from the input link.
F2 = Compare the packet received with the rules stated.
F3 = If it satisfies the rules then forward it on correct output link.
F4 = If the packet doesn't satisfy the rules then drop it.

SUCCESS = {All the packets that are passed satisfy the rules}

FAILURE = {Packets are forwarded even though they don't satisfy the rules}

Packet Content = {I, O, F, SUCCESS, FAILURE}
Where,
I = {E0, E1, E2…En}
[Ethernet Packets from the given interface]

O = {D0, D1, D2…Dn}
[Data related each packet]

F = {F1, F2, F3...Fn}
[Functions of Packet Content Application]

Where,
F1 = Defining offsets different fields in headers.

UPnP = {Host, Router} where,
Host = {I, O, F, SUCCESS, FAILURE}
Where,

I = {R1, R2...Rn, RS1, RS2...RS3}
[Host receives packets which are request from other hosts or response to its own request]

O = {R1, R2...Rn, RS1, RS2,.. RS3, RG1… RGn}

[Host is capable of sending requests and responses to requests and send registration packets to the router]

F = {F1, F2, F3....Fn} [Functions Of Host] Where,

F1 = Register itself to the router. F2 = Make a request packet to get the IP address of particular host.
F3 = Respond to a request sent by other host. F4 = Sending discovery packet F5 = Sending advertising packet

SUCCESS = {Registration is successfully done, Request is sent and response is received}

FAILURE = {The Response is not received even after sending the discovery packet}

Router = {I, O, F, SUCCESS, FAILURE}
Where,
I = {P1, P2, P3…Pn, RG1…RGn}
[Router receives packets which help the host register itself to the router and other packets are request/response which it simply forwards]

O = {P1, P2, P3...Pn} [Router forwards multicast request/response packets to other hosts]

F = {F1, F2, F3....Fn} [Functions of Router] Where,

F1 = Register the ip address of new host connected.

F2 = Forward the response to particular IP.

F3 = Multicast the request packet to other hosts on the network.

SUCCESS = {Registration is successfully done, Request is sent and response is received}

FAILURE = {The Response is not forwarded if the correct IP address is not present}

Port Forwarding = {I, O, F, SUCCESS, FAILURE}
where,
I = {P1, P2…Pn}
[Pi is an ethernet packet received with global IP address of router]

O = {O1, O2…On}
[Oi is an Ethernet packet with destination IP address of device within the network]

F = {F1, F2…Fn}
where,
F1 = Reading the destination port number and forwarding the packet to corresponding host on internal network.

SUCCESS = {All the packets received are forwarded successfully depending destination on the port numbers}

FAILURE = {Destination Host on internal not found corresponding to destination port number in the packet}

NIC (Network Interface Card)
A network interface controller (NIC, also known as a network interface card, network adapter, LAN adapter, and by similar terms) is a computer hardware component that connects a computer to a computer network (Ethernet type).

DD (Device Driver)
User space device driver which enables communication between NIC and virtual network stack.

Φ = {ATM, Token ring, Frame Relay incompatibility}

CONCLUSION

The framework is used to solve novel problems in networking. It provides different functionalities and interfaces to develop different networking functionalities and use them as applications. The framework provides mechanisms to use these different applications together to form a chain of applications. The interfaces provided by the framework are implemented to simulate the working of UPnP and mDNS protocols. The implementation simulates actual devices in the network by use of objects and uses links provided by the framework to transfer packets. The basic packet content application displays all the contents of the packet just like tcpdump command in linux. The basic firewall application filters packets based on the arguments passed to it. These are the four applications implemented using the Snabbswitch framework. Thus we conclude that development of network based applications has become flexible and easy task using Snabbswitch framework.

REFERENCES

[1] IEEE Paper:- TCP/IP protocol accelaration published in Computer Communication and Informatics (ICCCI), 2012 International Conference. http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?tp&arnumber=6158829&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6158829

[2] Virtual Network Stack:- Computer Science Department, University of Basel Switzerland http://conferences.sigcomm.org/sigcomm/2008/workshops/presto/papers/p45.pdf

[3] Snabbswitch Development http://www.snabb.co/

[4] Luke Gorrie – Snabbswitch Devoeloper http://lukego.github.io/

[5] Network Virtualization http://bradhedlund.com/2013/05/28/what-is-network-virtualization/

[6] Snabbswitch development Google group (forum) https://groups.google.com/forum/#!forum/snabb-devel