PROJECT REPORT

Computer Networks

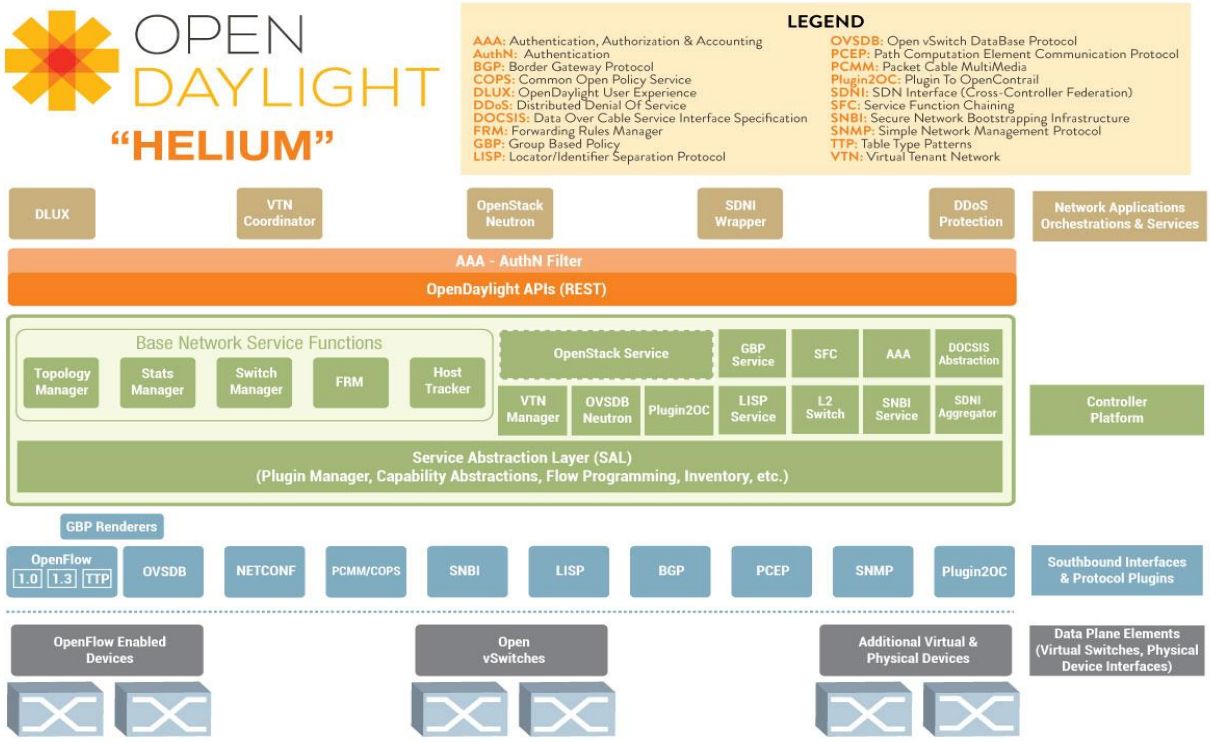
Shoaib Ahmed Siddiqui (00468)
Bilal Zahid (00450)
BSCS-2B

OpenDaylight

Introduction:

OpenDaylight is an open source platform for enabling SDN (Software Defined Networking) and NFV (Network Functions Virtualization). Software Defined Networking (SDN) separates the control plane from the data plane and allows the intelligence to be embedded as a centralized system. The centralized system communicates with the nodes i.e. switches and routers using OpenFlow protocol therefore only OpenFlow enabled devices can be used in SDN. OpenDaylight also has the capability of connecting with OpenStack. Software Defined Networks has the unique capability of adapting to the requirements.
We will now discuss each of the components of OpenDaylight in detail.

Basic Architecture:



Layers in OpenDaylight Architecture:

There are three layers in OpenDaylight architecture defined below:

1. **Network Applications & Orchestration**
   This is the top layer of the OpenDaylight architecture. It contains the business logic for managing the network as well as network applications for monitoring its behavior. A software defined network has the ability to adapt according to requirements.

2. **Controller Platform**
   Controller Platform is the heart of OpenDaylight. It has numerous APIs for interaction with the Application layer called as the "Northbound APIs". It also has APIs to interact with the original hardware to translate the design and topology specified by the network administrator as the "Southbound APIs".

3. **Physical and Virtual Network Devices**
   This is the bottom layer of the OpenDaylight architecture. It contains the routers and switches which actually needs to be controlled using the software.

Components of OpenDaylight:

There are many components of OpenDaylight. Few of them are defined below:

1. **Authentication Service**
   The authentication service is responsible for authenticating the users. A user can request authentication within a domain in which he has defined roles.

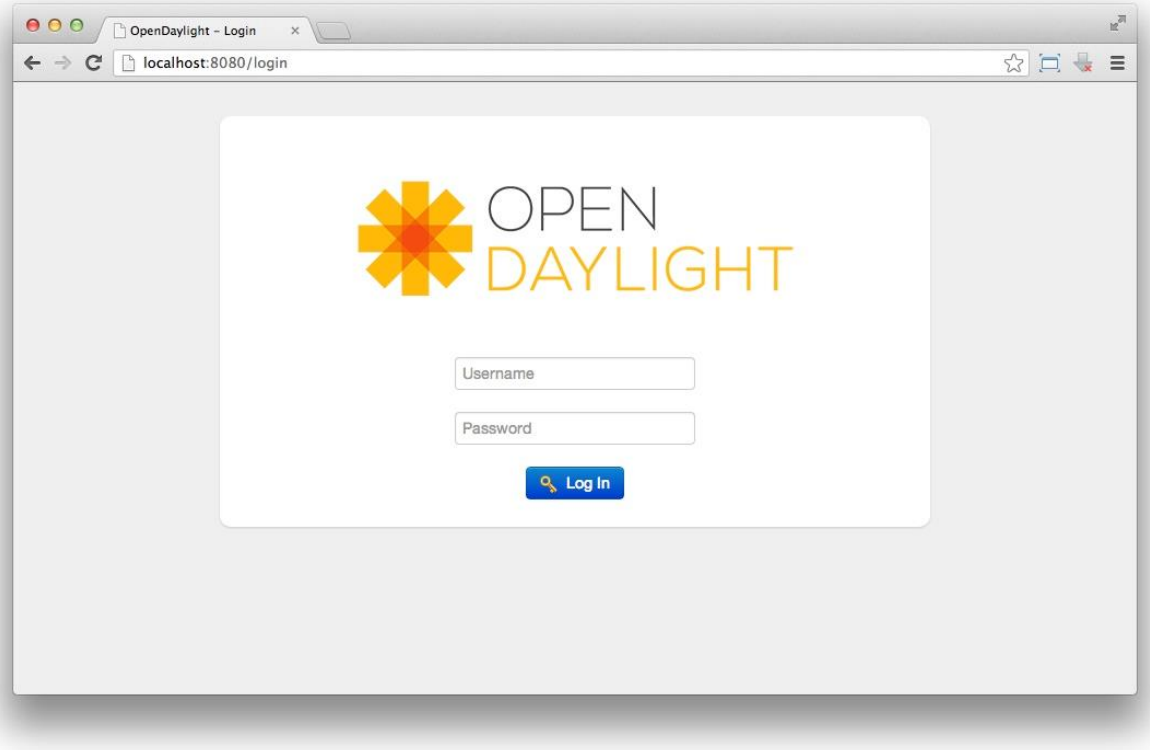
2. **Controller**
   Controller Platform is the heart of OpenDaylight. It has numerous APIs for interaction with the Application layer called as the "Northbound APIs". It also has APIs to interact with the original hardware to translate the design and topology specified by the network administrator as the "Southbound APIs".
   There are two types of SALs (Service Abstraction Layers):
   - **AD-SAL:**
     AD-SAL stands for API-Driven SAL. In AD-SAL, the API is defined at compile time and provider and consumer of events/data are directly plumbed into each other.
   - **MD-SAL:**
     MD-SAL stands for Model-Driven SAL. In MD-SAL, the API code is generated from models during compile time and loaded into the controller when the plugin OGSi is loaded.

3. Defense4All

Defense4All is a security application located at the top layer of OpenDaylight which provides protection against DDoS attacks. Administrator can configure Defense4All to protect certain networks or servers called as Protected Networks or Protected Objects.

4. DLUX

DLUX stands for "**D**ay**L**ight **U**ser **E**xperience". It was initially being developed from scratch but later, the developers realized that they were reinventing the wheel. OpenStack's Horizon already provided similar UI required by ODL therefore the developers finally adapted OpenStack's Horizon and modified it according to requirements which finally took DLUX's current form.



5. Group-Based Policy

Group-Based Policy component defines an application centric policy model for OpenDaylight that separates information about application connectivity requirements from information about the underlying details of the network infrastructure.

6. L2Switch

L2Switch component provides the functionality of a Layer 2 switch in OpenDaylight environment.

7. Lisp Flow Mapping

Locator/ID Separation Protocol (LISP) is a technology that provides a flexible map-and-encap framework that can be used for overlay network applications such as data center network virtualization and Network Function Virtualization (NFV).

LISP Flow Mapping Service store and serve mapping data to data plane nodes as well as to OpenDaylight applications.

8. ODL-SDNi

SDNi stands for Software Defined Networking Interface. It enables inter-SDN controller communication as an application.

9. OpenContrail Plugin

OpenContrail is an open-source network virtualization platform for the cloud.

This plugin provides the interaction between the OpenDaylight controller and OpenContrail platform. This allows OpenDaylight to use OpenContrail platform's networking capabilities.

10. OpenFlow Protocol Library

OpenFlow Protocol Library mediates communication between the OpenDaylight controller and the hardware devices supporting the OpenFlow protocol.

11. OpenFlow Plugin

OpenFlow is a standard communications interface defined to enable interactions between the control plane and the data plane.

12. OVSDB Integration

OVSDB stands for Open vSwitch Database. OVSDB Plugin component allows the southbound configuration of vSwitches. OVSDB protocol uses JSON/RPC calls to manipulate a physical or virtual switch that has OVSDB attached to it.

13. Virtual Tenant Network (VTN)

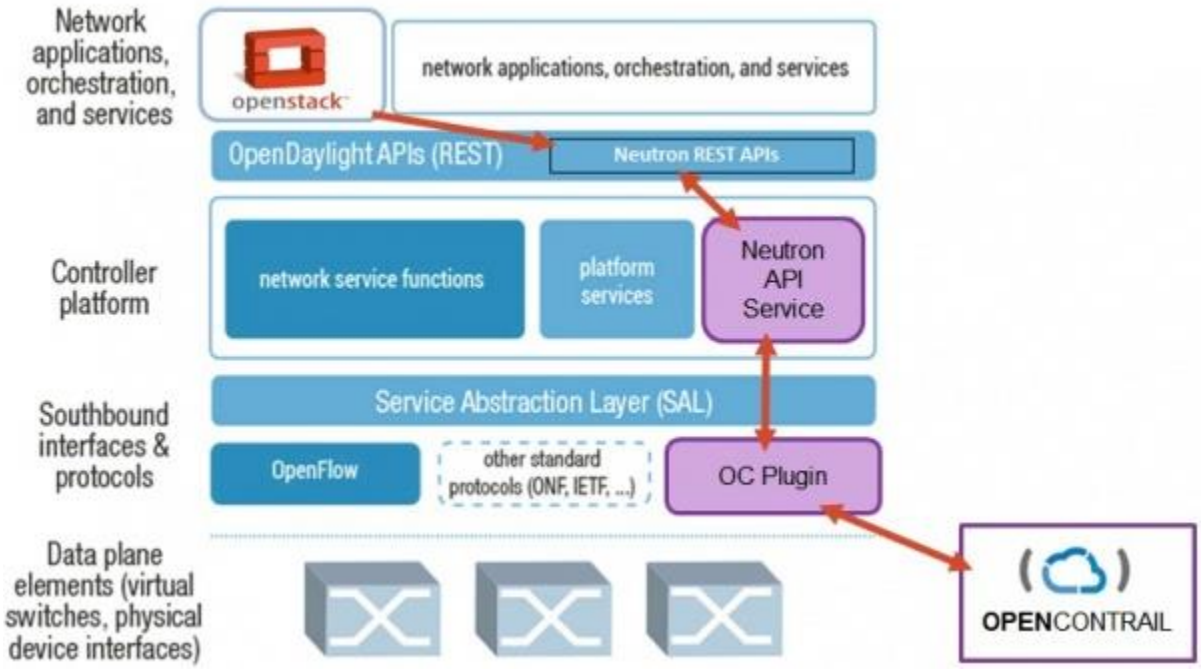
Virtual Tenant Network (VTN) is an application that provides multi-tenant virtual networks on an SDN controller.

VTN allows the user to define the network with a simple view hiding all the complexities of the underlying network. Once the network is designed onto VTN, it will automatically be mapped into the underlying physical network.

Integration with OpenStack:

OpenDaylight can be connected to the OpenStack's Neutron which is a module for providing networking as a service between interface devices. OpenDaylight will then take care of managing the underlying network hardware in order to reflect the changes made by the user in the network topologies. This will vastly reduce the cost of managing extremely complex networks at the data centers. Cloud is already a 420 billion $ industry and we need constructs like OpenDaylight for managing such a complex system which will vastly reduce its management costs.

OpenStack's Neutron communicates with the OpenDaylight controller via Neutron REST APIs available in the "Northbound APIs" of the controller platform. Neutron REST APIs communicates with the Neutron API service within the controller using SDNi which the interface for inter-controller communication. The Neutron API service in turn uses the services provided by the OpenContrail platform which provides network virtualization for the cloud.



Software Based Firewall (OpenDaylight Application)

Introduction:

Security has always been one of the prime concern for network managers. Hardware Firewalls are costly and needs to be configured separately. A good workaround could be to take advantage of the capabilities provided by OpenDaylight and implement a firewall in software. This will allow the network manager to impose rules on the network just by a simple web interface. The rules that can be imposed are:

- Block / Unblock Protocol
- Block / Unblock Port
- Block / Unblock IP Address
- Block / Unblock MAC Address

We will now define each of these functionalities in detail.

Functionalities:

Block / Unblock Protocol:

The network manager can block / unblock any protocol he wants just by providing the IP Protocol number. The protocol number for any protocol can be checked using the link given below:
http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

For example:
To block ICMP packets in the network, the network manager will click the block protocol button and enter the protocol number 1 which is the protocol number for ICMP. Hence the firewall will drop all the packets having protocol number 1.

Block / Unblock Port:

The network manager can block / unblock any port by specifying the port number to be blocked. If any packet contains that port as source or destination port, then the packet will be dropped immediately.

For example:
To block a Telnet communications within the network, the network manager will click block port button and enter 23 which is the port number used by Telnet. Hence the firewall will drop every packet having port 23 as source or destination.

Block / Unblock IP Address:

The network manager can block / unblock any IP Address by specifying the IP Address to be blocked along with its subnet mask. If any packet contains that IP Address as source or destination with the specified subnet, then the packet will be dropped immediately.

For example:
If an IP address “174.138.1.27” with subnet “255.255.255.0” is identified as malicious, then the network manager can block any traffic from or to that IP Address by clicking on the block IP Address button and entering the IP Address and subnet mask. Hence the firewall will drop every packet having the specified IP Address as source or destination along with the subnet mask.

Block / Unblock MAC Address:

The network manager can block / unblock any MAC Address by specifying the MAC Address to be blocked. If any packet contains that MAC Address as source or destination, then the packet will be dropped immediately.
For example:
If a MAC address “24-0A-3F- 6B-10- 05” is identified as malicious, then the network manager can block any traffic from or to that MAC Address by clicking on the block MAC Address button and entering the MAC Address. Hence the firewall will drop every packet having the specified MAC Address as source or destination.

Implementation:

TutorialL2Forwarding.java is a small file supplied with SDN Hub VM whose target is to create a learning switch. We modified the code in TutorialL2Forwarding.java to accommodate the needs of a firewall. Since it is an OpenDaylight application, we are using OpenDaylight’s DLUX for User Interface. There are several JSPs and XML mapping files written for that purpose. The source code for only the backend of the firewall is provided below:

Source Code:

TutorialL2Forwarding.java

```
/*
 * Copyright (C) 2014 SDN Hub

 Licensed under the GNU GENERAL PUBLIC LICENSE, Version 3.
 You may not use this file except in compliance with this License.
 You may obtain a copy of the License at

    http://www.gnu.org/licenses/gpl-3.0.txt

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 implied.

 *
 */
```

```
package org.opendaylight.tutorial.tutorial_L2_forwarding.internal;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.lang.String;
import java.util.Map;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.ConcurrentHashMap;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.framework.FrameworkUtil;
import org.opendaylight.controller.sal.core.ConstructionException;
import org.opendaylight.controller.sal.core.Node;
import org.opendaylight.controller.sal.core.NodeConnector;
import org.opendaylight.controller.sal.flowprogrammer.IFlowProgrammerService;
import org.opendaylight.controller.sal.flowprogrammer.Flow;
import org.opendaylight.controller.sal.packet.ARP;
import org.opendaylight.controller.sal.packet.BitBufferHelper;
import org.opendaylight.controller.sal.packet.Ethernet;
import org.opendaylight.controller.sal.packet.ICMP;
import org.opendaylight.controller.sal.packet.IDataPacketService;
import org.opendaylight.controller.sal.packet.IListenDataPacket;
import org.opendaylight.controller.sal.packet.IPv4;
import org.opendaylight.controller.sal.packet.LLDP;
import org.opendaylight.controller.sal.packet.Packet;
import org.opendaylight.controller.sal.packet.PacketResult;
import org.opendaylight.controller.sal.packet.RawPacket;
import org.opendaylight.controller.sal.packet.TCP;
import org.opendaylight.controller.sal.packet.UDP;
import org.opendaylight.controller.sal.action.Action;
import org.opendaylight.controller.sal.action.Drop;
import org.opendaylight.controller.sal.action.Output;
import org.opendaylight.controller.sal.action.Flood;
import org.opendaylight.controller.sal.match.Match;
import org.opendaylight.controller.sal.match.MatchType;
import org.opendaylight.controller.sal.match.MatchField;
import org.opendaylight.controller.sal.utils.EtherTypes;
import org.opendaylight.controller.sal.utils.Status;
import org.opendaylight.controller.sal.utils.NetUtils;
import org.opendaylight.controller.switchmanager.ISwitchManager;
import org.opendaylight.controller.switchmanager.Subnet;

public class TutorialL2Forwarding implements IListenDataPacket {
```

```
private static final Logger logger = LoggerFactory
      .getLogger(TutorialL2Forwarding.class);
private ISwitchManager switchManager = null;
private IFlowProgrammerService programmer = null;
private IDataPacketService dataPacketService = null;
private Map<Long, NodeConnector> mac_to_port = new HashMap<Long, NodeConnector>();
private String function = "switch";

//Contains all the nodes
private ArrayList<Node> nodes = new ArrayList<Node>();

//Firewall Attributes
//Blocked Ports
private ArrayList<Short> blockedPorts= new ArrayList<Short>();
//Blocked MACs
private ArrayList<Long> blockedMACs = new ArrayList<Long>();
//Blocked IPs (IP to Subnet Mapping)
private Map<InetAddress, InetAddress> blockedIPs = new HashMap<InetAddress, InetAddress>();
//Blocked Protocols
private ArrayList<Byte> blockedProtocols = new ArrayList<Byte>();

void setDataPacketService(IDataPacketService s) {
   this.dataPacketService = s;
}

void unsetDataPacketService(IDataPacketService s) {
   if (this.dataPacketService == s) {
      this.dataPacketService = null;
   }
}

public void setFlowProgrammerService(IFlowProgrammerService s)
{
   this.programmer = s;
}

public void unsetFlowProgrammerService(IFlowProgrammerService s) {
   if (this.programmer == s) {
      this.programmer = null;
   }
}

void setSwitchManager(ISwitchManager s) {
   logger.debug("SwitchManager set");
   this.switchManager = s;
}

void unsetSwitchManager(ISwitchManager s) {
   if (this.switchManager == s) {
      logger.debug("SwitchManager removed!");
      this.switchManager = null;
   }
}
```

```
/**
 * Function called by the dependency manager when all the required
 * dependencies are satisfied
 *
 */
void init() {
   logger.info("Initialized");
   // Disabling the SimpleForwarding and ARPHandler bundle to not conflict with this one
   BundleContext bundleContext = FrameworkUtil.getBundle(this.getClass()).getBundleContext();
   for(Bundle bundle : bundleContext.getBundles()) {
      if (bundle.getSymbolicName().contains("simpleforwarding")) {
         try {
            bundle.uninstall();
         } catch (BundleException e) {
            logger.error("Exception in Bundle uninstall "+bundle.getSymbolicName(), e);
         }
      }
   }

}

/**
 * Function called by the dependency manager when at least one
 * dependency become unsatisfied or when the component is shutting
 * down because for example bundle is being stopped.
 *
 */
void destroy() {
}

/**
 * Function called by dependency manager after "init ()" is called
 * and after the services provided by the class are registered in
 * the service registry
 *
 */
void start() {
   logger.info("Started");
}

/**
 * Function called by the dependency manager before the services
 * exported by the component are unregistered, this will be
 * followed by a "destroy ()" calls
 *
 */
void stop() {
   logger.info("Stopped");
}

private void floodPacket(RawPacket inPkt) {
   NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();
   Node incoming_node = incoming_connector.getNode();
```

```
    Set<NodeConnector> nodeConnectors =
        this.switchManager.getUpNodeConnectors(incoming_node);

    for (NodeConnector p : nodeConnectors) {
      if (!p.equals(incoming_connector)) {
        try {
          RawPacket destPkt = new RawPacket(inPkt);
          destPkt.setOutgoingNodeConnector(p);
          this.dataPacketService.transmitDataPacket(destPkt);
        } catch (ConstructionException e2) {
          continue;
        }
      }
    }
}


//Firewall initializer functions interacting with the UI
public void blockProtocol(byte protocol)
{
        blockedProtocols.add(protocol);
        resetNodes();
}

public void unblockProtocol(byte protocol)
{
        blockedProtocols.remove((Byte) protocol);
        resetNodes();
}

public void blockPort(short port)
{
        blockedPorts.add(port);
        resetNodes();
}

public void unblockPort(short port)
{
        blockedPorts.remove((Short) port);
        resetNodes();
}

public void blockIP(String ip, String sub)
{
        try
        {
                InetAddress IP = InetAddress.getByName(ip);
                InetAddress subnet = InetAddress.getByName(sub);

                blockedIPs.put(IP, subnet);
                resetNodes();
        }
        catch(Exception e)
        {
```

```
                e.printStackTrace();
        }
}

public void unblockIP(String ip)
{
        try
        {
                InetAddress IP = InetAddress.getByName(ip);
                blockedIPs.remove(IP);
                resetNodes();
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
}

public void blockMAC(long MAC)
{
        blockedMACs.add(MAC);
        resetNodes();
}

public void unblockMAC(long MAC)
{
        blockedMACs.remove((Long) MAC);
        resetNodes();
}

//Remove all flows from the router
public void resetNodes()
{
        for(Node node : nodes)
        {
                programmer.removeAllFlows(node);
        }
}



@Override
public PacketResult receiveDataPacket(RawPacket inPkt) {
   if (inPkt == null) {
      return PacketResult.IGNORED;
   }

   NodeConnector incoming_connector = inPkt.getIncomingNodeConnector();

   // Hub implementation
   if (function.equals("hub")) {
      floodPacket(inPkt);
   } else {
      Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);
```

```
if (!(formattedPak instanceof Ethernet)) {
   return PacketResult.IGNORED;
}

//Learn the node
learnNode(incoming_connector);

//Firewall Activities
//Check if protocol allowed
if(!checkIfProtocolAllowed(formattedPak))
{
      if(programFirewallRejectProtocolFlow(formattedPak, incoming_connector))
                              return PacketResult.CONSUME;
                  else
                              return PacketResult.IGNORED;
}
//Check if port allowed
if(!checkIfPortAllowed(formattedPak))
{
      if(programFirewallRejectPortFlow(formattedPak, incoming_connector))
                              return PacketResult.CONSUME;
                  else
                              return PacketResult.IGNORED;
}
//Check if IP allowed
if(!checkIfIPAllowed(formattedPak))
{
      if(programFirewallRejectIPFlow(formattedPak, incoming_connector))
                              return PacketResult.CONSUME;
                  else
                              return PacketResult.IGNORED;
}
//Check if MAC allowed
if(!checkIfMACAllowed(formattedPak))
{
      if(programFirewallRejectMACFlow(formattedPak, incoming_connector))
                  return PacketResult.CONSUME;
      else
                  return PacketResult.IGNORED;
}

//If not blocked by the firewall, program that flow
learnSourceMAC(formattedPak, incoming_connector);
NodeConnector outgoing_connector =
   knowDestinationMAC(formattedPak);

if (outgoing_connector == null) {
   floodPacket(inPkt);
} else {
   if (!programFlow(formattedPak, incoming_connector,
         outgoing_connector)) {
      return PacketResult.IGNORED;
   }
   inPkt.setOutgoingNodeConnector(outgoing_connector);
```

```
            this.dataPacketService.transmitDataPacket(inPkt);
        }
    }
    return PacketResult.CONSUME;
}

private void learnNode(NodeConnector incoming_connector)
{
        Node node = incoming_connector.getNode();
        if(!nodes.contains(node))
        {
                nodes.add(node);
        }
}

private void learnSourceMAC(Packet formattedPak, NodeConnector incoming_connector) {
    byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
    long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    this.mac_to_port.put(srcMAC_val, incoming_connector);
}

private NodeConnector knowDestinationMAC(Packet formattedPak) {
    byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);
    return this.mac_to_port.get(dstMAC_val);
}

//Block packet if of a blocked protocol
private boolean checkIfProtocolAllowed(Packet packet) {
        if(packet instanceof IPv4)
        {
                byte protocol = ((IPv4) packet).getProtocol();

                if(blockedProtocols.contains(protocol))
                {
                        return false;
                }
        }

        return true;
}

//Block traffic from or to this MAC address if found in MAC blockedlist
private boolean checkIfMACAllowed(Packet packet) {
        byte[] srcMAC = ((Ethernet)packet).getSourceMACAddress();
        byte[] dstMAC = ((Ethernet)packet).getDestinationMACAddress();

        long srcMAC_val = BitBufferHelper.toNumber(srcMAC);
    long dstMAC_val = BitBufferHelper.toNumber(dstMAC);

    if((blockedMACs.contains(dstMAC_val)) || (blockedMACs.contains(srcMAC_val)))
    {
        return false;
    }
```

```
		return true;
	}

	//Block traffic from or to this Port if found in Port blockedlist
	//Ports can only be checked with TCP and UDP
	private boolean checkIfPortAllowed(Packet packet) {
		if((packet instanceof TCP) || (packet instanceof UDP))
		{
			short srcPort = ((TCP) packet).getSourcePort();
			short dstPort = ((TCP) packet).getDestinationPort();

			if((blockedPorts.contains(srcPort)) || (blockedPorts.contains(dstPort)))
			{
				return false;
			}
		}

		return true;
	}

	//Block traffic from or to this IP address if found in IP blockedlist
	private boolean checkIfIPAllowed(Packet packet) {
		if(packet instanceof IPv4)
		{
			int srcIpAddr = ((IPv4) packet).getSourceAddress();
			int dstIpAddr = ((IPv4) packet).getDestinationAddress();

			String src = convertIPtoString(srcIpAddr);
			String dst = convertIPtoString(dstIpAddr);

			try
			{
				InetAddress srcIP = InetAddress.getByName(src);
				InetAddress dstIP = InetAddress.getByName(dst);

				if(blockedIPs.get(srcIP) != null)
				{
					//If source IP is blocked
					return false;
				}
				else if(blockedIPs.get(dstIP) != null)
				{
					//If destination IP is blocked
					return false;
				}
			}
			catch(Exception e)
			{
				e.printStackTrace();
			}
		}

		return true;
```

```
}

private String convertIPtoString(int ip)
{
        String ipStr =
         String.format("%d.%d.%d.%d",
              (ip & 0xff),
              (ip >> 8 & 0xff),
              (ip >> 16 & 0xff),
              (ip >> 24 & 0xff));
        return ipStr;
}

private boolean programFlow(Packet formattedPak,
    NodeConnector incoming_connector,
    NodeConnector outgoing_connector) {
  byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();

  Match match = new Match();
  match.setField( new MatchField(MatchType.IN_PORT, incoming_connector) );
  match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()) );

  List<Action> actions = new ArrayList<Action>();
  actions.add(new Output(outgoing_connector));

  Flow f = new Flow(match, actions);
  f.setIdleTimeout((short)5);

  // Modify the flow on the network node
  Node incoming_node = incoming_connector.getNode();
  Status status = programmer.addFlow(incoming_node, f);

  if (!status.isSuccess()) {
    logger.warn("SDN Plugin failed to program the flow: {}. The failure is: {}",
        f, status.getDescription());
    return false;
  } else {
    return true;
  }
}

private boolean programFirewallRejectProtocolFlow(Packet formattedPak,
                    NodeConnector incoming_connector) {
        if(formattedPak instanceof IPv4)
        {
            byte protocol = ((IPv4) formattedPak).getProtocol();

            Match match = new Match();
            match.setField( new MatchField(MatchType.NW_PROTO, protocol));

            List<Action> actions = new ArrayList<Action>();
            actions.add(new Drop());

            Flow f = new Flow(match, actions);
```

```
                f.setIdleTimeout((short)5);

                // Modify the flow on the network node
                Node incoming_node = incoming_connector.getNode();
                Status status = programmer.addFlow(incoming_node, f);

                if (!status.isSuccess()) {
                  logger.warn("SDN Plugin failed to program the reject protocol flow: {}. The failure is: {}",
                      f, status.getDescription());
                } else {
                  return true;
                }
            }

  return false;
}

private boolean programFirewallRejectPortFlow(Packet formattedPak,
      NodeConnector incoming_connector) {
            if((formattedPak instanceof TCP) || (formattedPak instanceof UDP))
            {
                        short srcPort;
                        short dstPort;

                        if(formattedPak instanceof TCP)
                        {
                                    srcPort = ((TCP) formattedPak).getSourcePort();
                                    dstPort = ((TCP) formattedPak).getDestinationPort();
                        }
                        else
                        {
                                    srcPort = ((UDP) formattedPak).getSourcePort();
                                    dstPort = ((UDP) formattedPak).getDestinationPort();
                        }

                Match match = new Match();
                if(blockedPorts.contains(srcPort))
                {
                        match.setField( new MatchField(MatchType.TP_SRC, srcPort));
                        match.setField( new MatchField(MatchType.TP_DST, srcPort));
                }
                else if(blockedPorts.contains(dstPort))
                {
                        match.setField( new MatchField(MatchType.TP_SRC, dstPort));
                        match.setField( new MatchField(MatchType.TP_DST, dstPort));
                }
                else
                {
                        return false;
                }

                List<Action> actions = new ArrayList<Action>();
                actions.add(new Drop());
```

```
            Flow f = new Flow(match, actions);
            f.setIdleTimeout((short)5);

            // Modify the flow on the network node
            Node incoming_node = incoming_connector.getNode();
            Status status = programmer.addFlow(incoming_node, f);

            if (!status.isSuccess()) {
              logger.warn("SDN Plugin failed to program the reject port flow: {}. The failure is: {}",
                  f, status.getDescription());
            } else {
              return true;
            }
          }

    return false;
}

private boolean programFirewallRejectIPFlow(Packet packet,
                    NodeConnector incoming_connector) {
          if(packet instanceof IPv4)
          {
                    int srcIpAddr = ((IPv4) packet).getSourceAddress();
                    int dstIpAddr = ((IPv4) packet).getDestinationAddress();

                    String src = convertIPtoString(srcIpAddr);
                    String dst = convertIPtoString(dstIpAddr);

                    try
                    {
                              InetAddress srcIP = InetAddress.getByName(src);
                              InetAddress dstIP = InetAddress.getByName(dst);

                        Match match = new Match();

                        InetAddress srcSub = blockedIPs.get(srcIP);
                        InetAddress dstSub = blockedIPs.get(dstIP);

                        if(srcSub != null)
                        {
                              match.setField( new MatchField(MatchType.NW_SRC, srcIP, srcSub));
                              match.setField( new MatchField(MatchType.NW_DST, srcIP, srcSub));
                        }
                        else if(dstSub != null)
                        {
                              match.setField( new MatchField(MatchType.NW_SRC, dstIP, dstSub));
                              match.setField( new MatchField(MatchType.NW_DST, dstIP, dstSub));
                        }
                        else
                        {
                              return false;
                        }

                        List<Action> actions = new ArrayList<Action>();
```

```
                        actions.add(new Drop());

                        Flow f = new Flow(match, actions);
                        f.setIdleTimeout((short)5);

                        // Modify the flow on the network node
                        Node incoming_node = incoming_connector.getNode();
                        Status status = programmer.addFlow(incoming_node, f);

                        if (!status.isSuccess()) {
                          logger.warn("SDN Plugin failed to program the reject IP flow: {}. The failure is: {}",
                              f, status.getDescription());
                        } else {
                          return true;
                        }
                    }
                    catch(Exception e)
                    {
                            e.printStackTrace();
                    }
        }

        return false;
}

private boolean programFirewallRejectMACFlow(Packet formattedPak,
                    NodeConnector incoming_connector) {
        byte[] srcMAC = ((Ethernet)formattedPak).getSourceMACAddress();
  byte[] dstMAC = ((Ethernet)formattedPak).getDestinationMACAddress();

  Match match = new Match();
  if(blockedMACs.contains(srcMAC))
  {
        match.setField( new MatchField(MatchType.DL_SRC, srcMAC.clone()));
        match.setField( new MatchField(MatchType.DL_DST, srcMAC.clone()));
  }
  else if(blockedMACs.contains(dstMAC))
  {
        match.setField( new MatchField(MatchType.DL_SRC, dstMAC.clone()));
        match.setField( new MatchField(MatchType.DL_DST, dstMAC.clone()));
  }
  else
  {
        return false;
  }

  List<Action> actions = new ArrayList<Action>();
  actions.add(new Drop());

  Flow f = new Flow(match, actions);
  f.setIdleTimeout((short)5);

  // Modify the flow on the network node
  Node incoming_node = incoming_connector.getNode();
```

```
        Status status = programmer.addFlow(incoming_node, f);

        if (!status.isSuccess()) {
          logger.warn("SDN Plugin failed to program the reject MAC flow: {}. The failure is: {}",
                f, status.getDescription());
          return false;
        } else {
          return true;
        }
      }
    }
```