**Learning switch source code**

**package** net.floodlightcontroller.mactracker;

**import** org.projectfloodlight.openflow.types.EthType;

**import** org.projectfloodlight.openflow.types.IPv4Address;

**import** org.projectfloodlight.openflow.types.IpProtocol;

**import** org.projectfloodlight.openflow.types.MacAddress;

**import** org.projectfloodlight.openflow.types.OFBufferId;

**import** org.projectfloodlight.openflow.types.OFPort;

**import** net.floodlightcontroller.topology.ITopologyService;

**import** net.floodlightcontroller.packet.ICMP;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Set;

**import** net.floodlightcontroller.core.IFloodlightProviderService;

**import** java.util.ArrayList;

**import** java.util.concurrent.ConcurrentSkipListSet;

**import** java.util.\*;

**import** net.floodlightcontroller.packet.Ethernet;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** java.util.Map;

**import** org.projectfloodlight.openflow.protocol.OFFlowMod;

**import** org.projectfloodlight.openflow.protocol.OFMessage;

**import** org.projectfloodlight.openflow.protocol.OFType;

**import** org.projectfloodlight.openflow.protocol.OFPacketIn;

**import** org.projectfloodlight.openflow.protocol.OFPacketOut;

**import** org.projectfloodlight.openflow.protocol.action.OFAction;

**import** org.projectfloodlight.openflow.protocol.OFFlowModCommand;

**import** net.floodlightcontroller.core.FloodlightContext;

**import** net.floodlightcontroller.core.IOFMessageListener;

**import** net.floodlightcontroller.core.IOFSwitch;

**import** net.floodlightcontroller.learningswitch.\*;

**import** net.floodlightcontroller.core.module.FloodlightModuleContext;

**import** net.floodlightcontroller.core.module.FloodlightModuleException;

**import** net.floodlightcontroller.core.module.IFloodlightModule;

**import** net.floodlightcontroller.core.module.IFloodlightService;

**import** org.projectfloodlight.openflow.protocol.match.Match;

**import** org.projectfloodlight.openflow.protocol.match.MatchField;

**public** **class** MACTracker **implements** IOFMessageListener, IFloodlightModule {

**protected** IFloodlightProviderService floodlightProvider;

**protected** Set<Long> macAddresses;

**protected** **static** Logger *logger*;

// more flow-mod defaults

**protected** **static** **short** *FLOWMOD\_DEFAULT\_IDLE\_TIMEOUT* = 5; // in seconds

**protected** **static** **short** *FLOWMOD\_DEFAULT\_HARD\_TIMEOUT* = 0; // infinite

**protected** **static** **short** *FLOWMOD\_PRIORITY* = 100;

**static** Map<String,Object> *mapping* = **new** HashMap<String, Object>();

**static** OFPort *OutPort*;

**static** **int** *a*=0,*b*=0;//control variable

//learning switch reverse flow

**protected** **static** **final** **boolean** ***LEARNING\_SWITCH\_REVERSE\_FLOW*** = **true**;

@Override

**public** String getName() {

// **TODO** Auto-generated method stub

**return** MACTracker.**class**.getSimpleName();

}

@Override

**public** **boolean** isCallbackOrderingPrereq(OFType type, String name) {

// **TODO** Auto-generated method stub

**return** **false**;

}

@Override

**public** **boolean** isCallbackOrderingPostreq(OFType type, String name) {

// **TODO** Auto-generated method stub

**return** **false**;

}

@Override

**public** Collection<Class<? **extends** IFloodlightService>> getModuleServices() {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** Map<Class<? **extends** IFloodlightService>, IFloodlightService> getServiceImpls() {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** Collection<Class<? **extends** IFloodlightService>> getModuleDependencies() {

// **TODO** Auto-generated method stub

Collection<Class<? **extends** IFloodlightService>> l = **new** ArrayList<Class<? **extends** IFloodlightService>>();

l.add(IFloodlightProviderService.**class**);

l.add(ITopologyService.**class**);

**return** l;

}

@Override

**public** **void** init(FloodlightModuleContext context)

**throws** FloodlightModuleException {

// **TODO** Auto-generated method stub

floodlightProvider = context.getServiceImpl(IFloodlightProviderService.**class**);

macAddresses = **new** ConcurrentSkipListSet<Long>();

*logger* = LoggerFactory.*getLogger*(MACTracker.**class**);

}

@Override

**public** **void** startUp(FloodlightModuleContext context)

**throws** FloodlightModuleException {

// **TODO** Auto-generated method stub

floodlightProvider.addOFMessageListener(OFType.***PACKET\_IN***, **this**);

}

@Override

**public** net.floodlightcontroller.core.IListener.Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {

// **TODO** Auto-generated method stub

Ethernet eth = IFloodlightProviderService.***bcStore***.get(cntx, IFloodlightProviderService.***CONTEXT\_PI\_PAYLOAD***);

OFPacketIn pi = (OFPacketIn)msg;

Map<String, Object> getmapping = **new** HashMap<String, Object>();

**if**(eth.getEtherType() == EthType.***ARP***) {

*logger*.info("ARP packet");

//every new MAC address

*a*=*b*;//control parameters to check if it finds the MAC:port information of the destination host

getmapping=addToPortMap(eth.getSourceMACAddress(),pi.getMatch().get(MatchField.***IN\_PORT***));

//the MAC Address of the host and the switchport that the host is on is added to the switch

SearchPort(getmapping,eth.getDestinationMACAddress());

**if**(*a*!=*b*) //if the mac address to port mapping already exists

{

MacAddress srcMac = eth.getSourceMACAddress();

MacAddress dstMac = eth.getDestinationMACAddress();

Match.Builder mb = sw.getOFFactory().buildMatch();

//Source and destination MAC address along with the source port is set in the match

mb.setExact(MatchField.***IN\_PORT***, pi.getMatch().get(MatchField.***IN\_PORT***))

.setExact(MatchField.***ETH\_SRC***, srcMac)

.setExact(MatchField.***ETH\_DST***, dstMac);

//build a packet to send to the switch in mininet

OFPacketOut.Builder pob = sw.getOFFactory().buildPacketOut();

List<OFAction> actions = **new** ArrayList<OFAction>();

actions.add(sw.getOFFactory().actions().buildOutput().setPort(*OutPort*).setMaxLen(0xffFFffFF).build());

pob.setActions(actions);

**if** (sw.getBuffers() == 0) {

// We set the PI buffer id here so we don't have to check again below

pi = pi.createBuilder().setBufferId(OFBufferId.***NO\_BUFFER***).build();

pob.setBufferId(OFBufferId.***NO\_BUFFER***);

} **else** {

pob.setBufferId(pi.getBufferId());

}

pob.setInPort(pi.getMatch().get(MatchField.***IN\_PORT***));

// If the buffer id is none or the switch doesn's support buffering

// we send the data with the packet out

**if** (pi.getBufferId() == OFBufferId.***NO\_BUFFER***) {

**byte**[] packetData = pi.getData();

pob.setData(packetData);

}

**if** (***LEARNING\_SWITCH\_REVERSE\_FLOW***)

{

Match.Builder mbr = sw.getOFFactory().buildMatch();

mbr.setExact(MatchField.***ETH\_SRC***, mb.get(MatchField.***ETH\_DST***))

.setExact(MatchField.***ETH\_DST***, mb.get(MatchField.***ETH\_SRC***))

.setExact(MatchField.***IN\_PORT***, *OutPort*);

**this**.writeFlowMod(sw, OFFlowModCommand.***ADD***, OFBufferId.***NO\_BUFFER***, mbr.build(), pi.getMatch().get(MatchField.***IN\_PORT***));

}

**this**.writeFlowMod(sw, OFFlowModCommand.***ADD***, OFBufferId.***NO\_BUFFER***, mb.build(), *OutPort*);

}

**else**

{

OFPort port = OFPort.***FLOOD***;

OFPacketOut.Builder pkout = sw.getOFFactory().buildPacketOut();

pkout.setBufferId(pi.getBufferId());

pkout.setXid(pi.getXid());

List<OFAction> actions = **new** ArrayList<OFAction>();

actions.add(sw.getOFFactory().actions().buildOutput().setPort(port).setMaxLen(0xffFFffFF).build());

pkout.setActions(actions);

**if** (pi.getBufferId() == OFBufferId.***NO\_BUFFER***) {

pkout.setData(pi.getData());

}

sw.write(pkout.build());

}

}

**return** Command.***CONTINUE***;

}

//This function creates a pairing of MAC address and the port number of the Source host

**public** Map<String, Object> addToPortMap(MacAddress mac, OFPort portVal) {

String srcPair=mac.toString();

**if**(!*mapping*.containsKey(srcPair)){

*mapping*.put(srcPair,portVal);

**return** *mapping*;

}

**else**{

System.***out***.println("Pair already included");

**return** *mapping*;

}

}

//search for the port number of the passed MAC address

**public** **void** SearchPort(Map<String,Object> display,MacAddress mac){

Set set=display.entrySet();

Iterator i=set.iterator();

String DestPair = mac.toString();

**while**(i.hasNext()){

Map.Entry me = (Map.Entry)i.next();

**if** (DestPair.equals(me.getKey())){

System.***out***.println("The port to send"+me.getValue());

*OutPort* = (OFPort)me.getValue();

System.***out***.println("Output port"+*OutPort*.toString());

*b*++; //increment b once we get the port value

}

**else**{

System.***out***.println("No port found");

}

}

}

//This is the main program that write a FlowMod to the switch

**private** **void** writeFlowMod(IOFSwitch sw, OFFlowModCommand command, OFBufferId bufferId,Match match, OFPort outPort){

OFFlowMod.Builder fmb;

**if** (command == OFFlowModCommand.***DELETE***) {

fmb = sw.getOFFactory().buildFlowDelete();

} **else** {

fmb = sw.getOFFactory().buildFlowAdd();

}

//Creating the action

fmb.setMatch(match);

fmb.setIdleTimeout(LearningSwitch.*FLOWMOD\_DEFAULT\_IDLE\_TIMEOUT*);

fmb.setHardTimeout(LearningSwitch.*FLOWMOD\_DEFAULT\_HARD\_TIMEOUT*);

fmb.setPriority(LearningSwitch.*FLOWMOD\_PRIORITY*);

fmb.setBufferId(bufferId);

fmb.setOutPort((command == OFFlowModCommand.***DELETE***) ? OFPort.***ANY*** : outPort);

List<OFAction> al = **new** ArrayList<OFAction>();

al.add(sw.getOFFactory().actions().buildOutput().setPort(outPort).setMaxLen(0xffFFffFF).build());

fmb.setActions(al);

**if** (*logger*.isTraceEnabled()) {

*logger*.trace("{} {} flow mod {}",

**new** Object[]{ sw, (command == OFFlowModCommand.***DELETE***) ? "deleting" : "adding", fmb.build() });

}

// and write it out

sw.write(fmb.build());

}

}