package org.sdnhub.odl.tutorial.learningswitch.impl;

import java.nio.ByteBuffer;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.opendaylight.controller.md.sal.binding.api.DataBroker;

import org.opendaylight.controller.md.sal.common.api.data.LogicalDatastoreType;

import org.opendaylight.controller.sal.binding.api.NotificationProviderService;

import org.opendaylight.controller.sal.binding.api.RpcProviderRegistry;

import org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.yang.types.rev100924.MacAddress;

import org.opendaylight.yang.gen.v1.urn.opendaylight.action.types.rev131112.action.list.Action;

import org.opendaylight.yang.gen.v1.urn.opendaylight.action.types.rev131112.action.action.OutputActionCaseBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.action.types.rev131112.action.action.output.action.\_case.OutputActionBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.action.types.rev131112.action.list.ActionBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.action.types.rev131112.action.list.ActionKey;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.FlowCapableNode;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.FlowId;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.tables.Table;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.tables.TableKey;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.tables.table.Flow;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.tables.table.FlowBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.tables.table.FlowKey;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.flow.InstructionsBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.flow.MatchBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.instruction.instruction.ApplyActionsCaseBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.instruction.instruction.apply.actions.\_case.ApplyActionsBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.instruction.list.Instruction;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.instruction.list.InstructionBuilder;

import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.types.rev131026.instruction.list.InstructionKey;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeConnectorId;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeConnectorRef;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeId;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeRef;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.Nodes;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.nodes.Node;

import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.nodes.NodeKey;

import org.opendaylight.yang.gen.v1.urn.opendaylight.packet.service.rev130709.PacketProcessingListener;

import org.opendaylight.yang.gen.v1.urn.opendaylight.packet.service.rev130709.PacketProcessingService;

import org.opendaylight.yang.gen.v1.urn.opendaylight.packet.service.rev130709.PacketReceived;

import org.opendaylight.yang.gen.v1.urn.opendaylight.packet.service.rev130709.TransmitPacketInput;

import org.opendaylight.yang.gen.v1.urn.opendaylight.packet.service.rev130709.TransmitPacketInputBuilder;

import org.opendaylight.yangtools.concepts.Registration;

import org.opendaylight.yangtools.yang.binding.InstanceIdentifier;

import org.sdnhub.odl.tutorial.utils.GenericTransactionUtils;

import org.sdnhub.odl.tutorial.utils.PacketParsingUtils;

import org.sdnhub.odl.tutorial.utils.inventory.InventoryUtils;

import org.sdnhub.odl.tutorial.utils.openflow13.MatchUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

public class TutorialL2Forwarding implements AutoCloseable, PacketProcessingListener {

private final Logger LOG = LoggerFactory.getLogger(this.getClass());

private final static long FLOOD\_PORT\_NUMBER = 0xfffffffbL;

//Members specific to this class

private Map<String, NodeConnectorId> macTable = new HashMap <String, NodeConnectorId>();

private String function = "hub";

//Members related to MD-SAL operations

private List<Registration> registrations;

private DataBroker dataBroker;

private PacketProcessingService packetProcessingService;

public TutorialL2Forwarding(DataBroker dataBroker, NotificationProviderService notificationService, RpcProviderRegistry rpcProviderRegistry) {

//Store the data broker for reading/writing from inventory store

this.dataBroker = dataBroker;

//Get access to the packet processing service for making RPC calls later

this.packetProcessingService = rpcProviderRegistry.getRpcService(PacketProcessingService.class);

//List used to track notification (both data change and YANG-defined) listener registrations

this.registrations = Lists.newArrayList();

//Register this object for receiving notifications when there are PACKET\_INs

registrations.add(notificationService.registerNotificationListener(this));

}

@Override

public void close() throws Exception {

for (Registration registration : registrations) {

registration.close();

}

registrations.clear();

}

@Override

public void onPacketReceived(PacketReceived notification) {

LOG.trace("Received packet notification {}", notification.getMatch());

NodeConnectorRef ingressNodeConnectorRef = notification.getIngress();

NodeRef ingressNodeRef = InventoryUtils.getNodeRef(ingressNodeConnectorRef);

NodeConnectorId ingressNodeConnectorId = InventoryUtils.getNodeConnectorId(ingressNodeConnectorRef);

NodeId ingressNodeId = InventoryUtils.getNodeId(ingressNodeConnectorRef);

// Useful to create it beforehand

NodeConnectorId floodNodeConnectorId = InventoryUtils.getNodeConnectorId(ingressNodeId, FLOOD\_PORT\_NUMBER);

NodeConnectorRef floodNodeConnectorRef = InventoryUtils.getNodeConnectorRef(floodNodeConnectorId);

/\*

\* Logic:

\* 0. Ignore LLDP packets

\* 1. If behaving as "hub", perform a PACKET\_OUT with FLOOD action

\* 2. Else if behaving as "learning switch",

\* 2.1. Extract MAC addresses

\* 2.2. Update MAC table with source MAC address

\* 2.3. Lookup in MAC table for the target node connector of dst\_mac

\* 2.3.1 If found,

\* 2.3.1.1 perform FLOW\_MOD for that dst\_mac through the target node connector

\* 2.3.1.2 perform PACKET\_OUT of this packet to target node connector

\* 2.3.2 If not found, perform a PACKET\_OUT with FLOOD action

\*/

//Ignore LLDP packets, or you will be in big trouble

byte[] etherTypeRaw = PacketParsingUtils.extractEtherType(notification.getPayload());

int etherType = (0x0000ffff & ByteBuffer.wrap(etherTypeRaw).getShort());

if (etherType == 0x88cc) {

return;

}

// Hub implementation

if (function.equals("hub")) {

//flood packet (1)

packetOut(ingressNodeRef, floodNodeConnectorRef, notification.getPayload());

} else {

byte[] payload = notification.getPayload();

byte[] dstMacRaw = PacketParsingUtils.extractDstMac(payload);

byte[] srcMacRaw = PacketParsingUtils.extractSrcMac(payload);

//Extract MAC addresses (2.1)

String srcMac = PacketParsingUtils.rawMacToString(srcMacRaw);

String dstMac = PacketParsingUtils.rawMacToString(dstMacRaw);

//Learn source MAC address (2.2)

this.macTable.put(srcMac, ingressNodeConnectorId);

//Lookup destination MAC address in table (2.3)

NodeConnectorId egressNodeConnectorId = this.macTable.get(dstMac) ;

//If found (2.3.1)

if (egressNodeConnectorId != null) {

programL2Flow(ingressNodeId, dstMac, ingressNodeConnectorId, egressNodeConnectorId);

NodeConnectorRef egressNodeConnectorRef = InventoryUtils.getNodeConnectorRef(egressNodeConnectorId);

packetOut(ingressNodeRef, egressNodeConnectorRef, payload);

} else {

//2.3.2 Flood packet

packetOut(ingressNodeRef, floodNodeConnectorRef, payload);

}

}

}

private void packetOut(NodeRef egressNodeRef, NodeConnectorRef egressNodeConnectorRef, byte[] payload) {

Preconditions.checkNotNull(packetProcessingService);

LOG.debug("Flooding packet of size {} out of port {}", payload.length, egressNodeConnectorRef);

//Construct input for RPC call to packet processing service

TransmitPacketInput input = new TransmitPacketInputBuilder()

.setPayload(payload)

.setNode(egressNodeRef)

.setEgress(egressNodeConnectorRef)

.build();

packetProcessingService.transmitPacket(input);

}

private void programL2Flow(NodeId nodeId, String dstMac, NodeConnectorId ingressNodeConnectorId, NodeConnectorId egressNodeConnectorId) {

/\* Programming a flow involves:

\* 1. Creating a Flow object that has a match and a list of instructions,

\* 2. Adding Flow object as an augmentation to the Node object in the inventory.

\* 3. FlowProgrammer module of OpenFlowPlugin will pick up this data change and eventually program the switch.

\*/

//Creating match object

MatchBuilder matchBuilder = new MatchBuilder();

MatchUtils.createEthDstMatch(matchBuilder, new MacAddress(dstMac), null);

MatchUtils.createInPortMatch(matchBuilder, ingressNodeConnectorId);

// Instructions List Stores Individual Instructions

InstructionsBuilder isb = new InstructionsBuilder();

List<Instruction> instructions = Lists.newArrayList();

InstructionBuilder ib = new InstructionBuilder();

ApplyActionsBuilder aab = new ApplyActionsBuilder();

ActionBuilder ab = new ActionBuilder();

List<Action> actionList = Lists.newArrayList();

// Set output action

OutputActionBuilder output = new OutputActionBuilder();

output.setOutputNodeConnector(egressNodeConnectorId);

output.setMaxLength(65535); //Send full packet and No buffer

ab.setAction(new OutputActionCaseBuilder().setOutputAction(output.build()).build());

ab.setOrder(0);

ab.setKey(new ActionKey(0));

actionList.add(ab.build());

// Create Apply Actions Instruction

aab.setAction(actionList);

ib.setInstruction(new ApplyActionsCaseBuilder().setApplyActions(aab.build()).build());

ib.setOrder(0);

ib.setKey(new InstructionKey(0));

instructions.add(ib.build());

// Create Flow

FlowBuilder flowBuilder = new FlowBuilder();

flowBuilder.setMatch(matchBuilder.build());

String flowId = "L2\_Rule\_" + dstMac;

flowBuilder.setId(new FlowId(flowId));

FlowKey key = new FlowKey(new FlowId(flowId));

flowBuilder.setBarrier(true);

flowBuilder.setTableId((short)0);

flowBuilder.setKey(key);

flowBuilder.setPriority(32768);

flowBuilder.setFlowName(flowId);

flowBuilder.setHardTimeout(0);

flowBuilder.setIdleTimeout(0);

flowBuilder.setInstructions(isb.setInstruction(instructions).build());

InstanceIdentifier<Flow> flowIID = InstanceIdentifier.builder(Nodes.class)

.child(Node.class, new NodeKey(nodeId))

.augmentation(FlowCapableNode.class)

.child(Table.class, new TableKey(flowBuilder.getTableId()))

.child(Flow.class, flowBuilder.getKey())

.build();

GenericTransactionUtils.writeData(dataBroker, LogicalDatastoreType.CONFIGURATION, flowIID, flowBuilder.build(), true);

}

}