APPENDIX

Blacklist Application

B

The source code included in this appendix may be downloaded from: www.tallac.com/SDN/get-started/.

B.1 MESSAGELISTENER

```
//=================================================================
//   MessageListener class for receiving openflow messages
//   from Floodlight controller.
//=================================================================

public class MessageListener implements IOFMessageListener
{
    private static final MessageListener INSTANCE =
                                            new MessageListener();

    private static IFloodlightProviderService mProvider;

    private MessageListener() {}  // private constructor

    public static MessageListener getInstance() {
        return INSTANCE;
    }

    //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    public void init(final FloodlightModuleContext context)
    {
        if( mProvider != null ) throw new RuntimeException(
                "BlackList Message listener already initialized" );

        mProvider = context.getServiceImpl(
                                IFloodlightProviderService.class );
    }
```

```
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void startUp()
{
    // Register class as MessageListener for PACKET_IN messages.
    mProvider.addOFMessageListener( OFType.PACKET_IN, this );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
@Override
public String getName() { return BlackListModule.NAME; }

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
@Override
public boolean isCallbackOrderingPrereq( final OFType type,
                                         final String name )
{
    return( type.equals( OFType.PACKET_IN ) &&
            ( name.equals("topology") ||
              name.equals("devicemanager") ) );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
@Override
public boolean isCallbackOrderingPostreq( final OFType type,
                                          final String name )
{
    return( type.equals( OFType.PACKET_IN ) &&
            name.equals( "forwarding" ) );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
@Override
public Command receive( final IOFSwitch         ofSwitch,
                        final OFMessage         msg,
                        final FloodlightContext context )
{
    switch( msg.getType() )
    {
    case PACKET_IN:  // Handle incoming packets here

       // Create packethandler object for receiving packet in
       PacketHandler ph = new PacketHandler( ofSwitch,
                                             msg, context);

       // Invoke processPacket() method of our packet handler
       // and return the value returned to us by processPacket
       return ph.processPacket();
```

```
        default: break;  // If not a PACKET_IN, just return

        }

        return Command.CONTINUE;
    }

}
```

B.2 PACKETHANDLER

```
//=================================================================
//   PacketHandler class for processing packets receives
//   from Floodlight controller.
//=================================================================

public class PacketHandler
{
    public static final short   TYPE_IPv4 = 0x0800;
    public static final short   TYPE_8021Q = (short) 0x8100;

    private final IOFSwitch         mOfSwitch;
    private final OFPacketIn        mPacketIn;
    private final FloodlightContext mContext;
    private       boolean           isDnsPacket;

    //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    public PacketHandler( final IOFSwitch         ofSwitch,
                          final OFMessage         msg,
                          final FloodlightContext context )
    {
        mOfSwitch   = ofSwitch;
        mPacketIn   = (OFPacketIn) msg;
        mContext    = context;
        isDnsPacket = false;
    }

    //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    public Command processPacket()
    {
        // First, get the OFMatch object from the incoming packet
        final OFMatch ofMatch = new OFMatch();
        ofMatch.loadFromPacket( mPacketIn.getPacketData(),
                                mPacketIn.getInPort()     );

        // If the packet isn't IPv4, ignore.
        if( ofMatch.getDataLayerType() != Ethernet.TYPE_IPv4 )
        {
            return Command.CONTINUE;
```

```
}

//—- First handle all IP packets _ _ _ _ _ _ _ _ _
// We have an IPv4 packet, so check the
// destination IPv4 address against IPv4 blacklist.
try
{
    // Get the IP address
    InetAddress ipAddr = InetAddress.getByAddress(
              IPv4.toIPv4AddressBytes(
                    ofMatch.getNetworkDestination() ) );

    // Check the IP address against our blacklist
    if( BlacklistMgr.getInstance()
                        .checkIpv4Blacklist( ipAddr ) )
    {
        // It's on the blacklist, so update stats...
        StatisticsMgr.getInstance()
                             .updateIpv4Stats( mOfSwitch,
                                               ofMatch,
                                               ipAddr    );

        // ... and drop the packet so it doesn't
        // go through to the destination.
        FlowMgr.getInstance().dropPacket( mOfSwitch,
                                          mContext,
                                          mPacketIn );

        return Command.STOP; // Done with this packet,
                             // don't let somebody else
                             // change our DROP
    }

}

catch( UnknownHostException e1 )
{
    // If we had an error with something, bad IP or some
    return Command.CONTINUE;
}

//—- Now handle DNS packets _ _ _ _ _ _ _ _

// Is it DNS?
if( ofMatch.getNetworkProtocol() == IPv4.PROTOCOL_UDP &&
    ofMatch.getTransportDestination()
                           == FlowMgr.DNS_QUERY_DEST_PORT )
{
```

```
// Prepare data structure to hold DNS hostnames
// we extract from the request
final byte[] pkt = mPacketIn.getPacketData();
Collection<String> domainNames;

isDnsPacket = true;

// Get the domain names from the DNS request
try
{
    domainNames = parseDnsPacket( pkt );
}
catch( IOException e )  // Got here if there was an
                        // exception in parsing the
                        // domain names.
{
    return Command.CONTINUE; // Just return and
                             // allow other apps
                             // to handle the
                             // request.
}

// If there were not any domain names,
// no checking required, just forward the
// packet and return.
if( domainNames == null ) {
    forwardPacket();
    return Command.STOP;
}

// If there are domain names, proces them.
for( String domainName : domainNames )
{
    //  If the current domainName is in
    //  the blacklist, drop the packet and return.
    if( BlacklistMgr.getInstance()
                    .checkDnsBlacklist( domainName ) )
    {
        // Update statistics about the dropped packet.
        StatisticsMgr.getInstance()
                         .updateDnsStats( mOfSwitch,
                                          ofMatch,
                                          domainName );

        // Drop the packet.
        FlowMgr.getInstance().dropPacket( mOfSwitch,
                                          mContext,
                                          mPacketIn );
```

```
                return Command.STOP; // Note that we are
                                     // dropping the whole
                                     // DNS request, even if
                                     // only one hostname is bad.
            }
        }
    }

    // If we made it here, everything is okay, so call the
    // method to forward the packet and set up flows for
    // the IP destination, if appropriate.
    forwardPacket();
    return Command.STOP;
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void forwardPacket()
{
    // Get the output port for this destination IP address.
    short outputPort = FlowMgr.getInstance()
              .getOutputPort( mOfSwitch, mContext, mPacketIn );

    // If we can't get a valid output port for this
    // destination IP address, we have to drop it.
    if( outputPort == OFPort.OFPP_NONE.getValue() )
              FlowMgr.getInstance().dropPacket( mOfSwitch,
                                                mContext,
                                                mPacketIn );

    // Else if we should flood the packet, do so.
    else if( outputPort == OFPort.OFPP_FLOOD.getValue() ) {
              FlowMgr.getInstance().floodPacket( mOfSwitch,
                                                 mContext,
                                                 mPacketIn );
    }

    // Else we have a port to send this packet out on, so do it.
    else
    {
        final List<OFAction> actions = new ArrayList<OFAction>();

        // Add the action for forward the packet out outputPort
        actions.add( new OFActionOutput( outputPort ) );

        // Note that for DNS requests,
        // we don't need to set flows up on the switch.
        // Otherwise we must set flows so that subsequent
```

```
            // packets to this IP dest will get forwarded
            // locally w/o the controller.
            if( !isDnsPacket )
               FlowMgr.getInstance().createDataStreamFlow( mOfSwitch,
                                                           mContext,
                                                           mPacketIn,
                                                           actions );

            // In all cases, we have the switch forward the packet.
            FlowMgr.getInstance().sendPacketOut( mOfSwitch,
                                                 mContext,
                                                 mPacketIn,
                                                 actions    );

        }

    }

    //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    private Collection<String> parseDnsPacket(byte[] pkt) throws IOException
    {
        // Code to parse DNS are return
        // a collection of hostnames
        // that were in the DNS request
    }

}
```

B.3 FLOWMANAGER

```
//=============================================================
//   FlowManager class for handling interactions with Floodlight
//   regarding setting and unsetting flows
//=============================================================

public class FlowMgr
{
    private static final FlowMgr INSTANCE = new FlowMgr();

    private static IFloodlightProviderService mProvider;
    private static ITopologyService mTopology;

    public static final short PRIORITY_NORMAL      = 10;
    public static final short PRIORITY_IP_PACKETS  = 1000;
    public static final short PRIORITY_DNS_PACKETS = 2000;
    public static final short PRIORITY_IP_FLOWS    = 1500;
```

```
public static final short PRIORITY_ARP_PACKETS = 1500;

public static final short IP_FLOW_IDLE_TIMEOUT = 15;
public static final short NO_IDLE_TIMEOUT = 0;
public static final int BUFFER_ID_NONE = 0xffffffff;

public static final short DNS_QUERY_DEST_PORT = 53;

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private FlowMgr()
{
    // private constructor - prevent external instantiation
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public static FlowMgr getInstance()
{
    return INSTANCE;
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void init( final FloodlightModuleContext context )
{
    mProvider = context.getServiceImpl(IFloodlightProviderService.class);
    mTopology = context.getServiceImpl(ITopologyService.class);
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void setDefaultFlows(final IOFSwitch ofSwitch)
{
    // Note: this method is called whenever a switch is
    //     discovered by Floodlight, in our SwitchListener
    //     class (not included in this appendix).

    // Set the intitial 'static' or 'proactive' flows
    setDnsQueryFlow(ofSwitch);
    setIpFlow(ofSwitch);
    setArpFlow(ofSwitch);
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void sendPacketOut( final IOFSwitch         ofSwitch,
                           final FloodlightContext cntx,
                           final OFPacketIn        packetIn,
                           final List<OFAction>    actions)
{
    // Create a packet out from factory.
    final OFPacketOut packetOut = (OFPacketOut) mProvider
```

```
                                .getOFMessageFactory()
                                .getMessage(OFType.PACKET_OUT);

    // Set the actions based on what has been passed to us.
    packetOut.setActions(actions);

    // Calculate and set the action length.
    int actionsLength = 0;
    for (final OFAction action : actions)
        { actionsLength += action.getLengthU(); }
    packetOut.setActionsLength((short) actionsLength);

    // Set the length based on what we've calculated.
    short poLength = (short) (packetOut.getActionsLength()
                              + OFPacketOut.MINIMUM_LENGTH);

    // Set the buffer and in port based on the packet in.
    packetOut.setBufferId( packetIn.getBufferId() );
    packetOut.setInPort(   packetIn.getInPort() );

    // If the buffer ID is not present, copy and send back
    // the complete packet including payload to the switch.
    if (packetIn.getBufferId() == OFPacketOut.BUFFER_ID_NONE)
    {
        final byte[] packetData = packetIn.getPacketData();
        poLength += packetData.length;
        packetOut.setPacketData(packetData);
    }

    // Set the complete length of the packet we are sending.
    packetOut.setLength( poLength );

    // Now we actually send out the packet.
    try
    {
        ofSwitch.write( packetOut, cntx );
        ofSwitch.flush();
    }
    catch (final IOException e)
    {
        // Handle errors in sending packet out.
    }
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void dropPacket( final IOFSwitch         ofSwitch,
                        final FloodlightContext cntx,
                              OFPacketIn        packetIn)
```

```
{
    LOG.debug("Drop packet");

    final List<OFAction> flActions = new ArrayList<OFAction>();
    sendPacketOut( ofSwitch, cntx, packetIn, flActions );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void createDataStreamFlow(
                          final IOFSwitch         ofSwitch,
                          final FloodlightContext context,
                          final OFPacketIn        packetIn,
                                List<OFAction>    actions)
{
    final OFMatch match = new OFMatch();
    match.loadFromPacket( packetIn.getPacketData(),
                          packetIn.getInPort() );

    // Ignore packet if it is an ARP, or has not source/dest, or is not IPv4.
    if( ( match.getDataLayerType() == Ethernet.TYPE_ARP) ||
        ( match.getNetworkDestination() == 0 )           ||
        ( match.getNetworkSource() == 0 )                ||
        ( match.getDataLayerType() != Ethernet.TYPE_IPv4 ) )
       return;

    // Set up the wildcard object for IP address.
    match.setWildcards( allExclude( OFMatch.OFPFW_NW_DST_MASK,
                                    OFMatch.OFPFW_DL_TYPE ) );

    // Send out the data stream flow mod message
    sendFlowModMessage( ofSwitch,
                        OFFlowMod.OFPFC_ADD,
                        match,
                        actions,
                        PRIORITY_IP_FLOWS,
                        IP_FLOW_IDLE_TIMEOUT,
                        packetIn.getBufferId() );

}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void deleteFlow( final IOFSwitch ofSwitch,
                         final OFMatch   match )
{
    // Remember that an empty action list means 'drop'
    final List<OFAction> actions = new ArrayList<OFAction>();

    // Send out our empty action list.
```

```
    sendFlowModMessage( ofSwitch,
                        OFFlowMod.OFPFC_DELETE,
                        match,
                        actions,
                        PRIORITY_IP_FLOWS,
                        IP_FLOW_IDLE_TIMEOUT,
                        BUFFER_ID_NONE );
}



//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public void floodPacket(final IOFSwitch ofSwitch,
                        final FloodlightContext context,
                        final OFPacketIn packetIn)
{
    // Create action flood/all
    final List<OFAction> actions = new ArrayList<OFAction>();

    // If the switch supports the 'FLOOD' action...
    if (ofSwitch.hasAttribute( IOFSwitch.PROP_SUPPORTS_OFPP_FLOOD) )
    {
        actions.add(new OFActionOutput(
                                OFPort.OFPP_FLOOD.getValue()));
    }
    // ...otherwise tell it to send it to 'ALL'.
    else
    {
        actions.add( new OFActionOutput(
                                OFPort.OFPP_ALL.getValue() ) );
    }

    // Call our method to send the packet out.
    sendPacketOut( ofSwitch,
                   context,
                   packetIn,
                   actions );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public short getOutputPort( final IOFSwitch ofSwitch,
                            final FloodlightContext context,
                            final OFPacketIn packetIn )
{
    // return the output port for sending out the packet
    // that has been approved
}
```

```
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private static int allExclude(final int... flags)
{
    // Utility routine for assistance in setting wildcard

    int wildcard = OFPFW_ALL;
    for( final int flag : flags ) { wildcard &= ~flag; }

    return wildcard;
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void sendFlowModMessage( final IOFSwitch      ofSwitch,
                                 final short          command,
                                 final OFMatch        ofMatch,
                                 final List<OFAction> actions,
                                 final short          priority,
                                 final short          idleTimeout,
                                 final int            bufferId )
{
    // Get a flow modification message from factory.
    final OFFlowMod ofm = (OFFlowMod) mProvider
                                 .getOFMessageFactory()
                                 .getMessage(OFType.FLOW_MOD);

    // Set our new flow mod object with the values that have
    // been passed to us.
    ofm.setCommand( command ).setIdleTimeout( idleTimeout )
                             .setPriority( priority )
                             .setMatch( ofMatch.clone() )
                             .setBufferId( bufferId )
                             .setOutPort( OFPort.OFPP_NONE )
                             .setActions( actions )
                             .setXid( ofSwitch
                                .getNextTransactionId() );

    // Calculate the length of the request, and set it.
    int actionsLength = 0;
    for( final OFAction action : actions ) { actionsLength += action.getLengthU(); }
    ofm.setLengthU(OFFlowMod.MINIMUM_LENGTH + actionsLength);

    // Now send out the flow mod message we have created.
    try
    {
        ofSwitch.write( ofm, null );
        ofSwitch.flush();
    }
    catch (final IOException e)
```

```
    {
        // Handle errors with the request
    }
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void setDnsQueryFlow( final IOFSwitch ofSwitch )
{
    // Create match object to only match DNS requests
    OFMatch ofMatch = new OFMatch();
    ofMatch.setWildcards( allExclude( OFPFW_TP_DST,
                                      OFPFW_NW_PROTO,
                                      OFPFW_DL_TYPE ) )
                .setDataLayerType( Ethernet.TYPE_IPv4 )
                .setNetworkProtocol( IPv4.PROTOCOL_UDP )
                .setTransportDestination( DNS_QUERY_DEST_PORT );

    // Create output action to forward to controller.
    OFActionOutput ofAction  = new OFActionOutput(
                              OFPort.OFPP_CONTROLLER.getValue(),
                              (short) 65535                    );

    // Create our action list and add this action to it
    List<OFAction> ofActions = new ArrayList<OFAction>();
    ofActions.add(ofAction);

    sendFlowModMessage( ofSwitch,
                        OFFlowMod.OFPFC_ADD,
                        ofMatch,
                        ofActions,
                        PRIORITY_DNS_PACKETS,
                        NO_IDLE_TIMEOUT,
                        BUFFER_ID_NONE );
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
private void setIpFlow( final IOFSwitch ofSwitch )
{
    // Create match object to only match all IPv4 packets
    OFMatch ofMatch = new OFMatch();
    ofMatch.setWildcards( allExclude( OFPFW_DL_TYPE ) )
           .setDataLayerType( Ethernet.TYPE_IPv4 );

    // Create output action to forward to controller.
    OFActionOutput ofAction  = new OFActionOutput(
                              OFPort.OFPP_CONTROLLER.getValue(),
                              (short) 65535                    );
```

```
        // Create our action list and add this action to it.
        List<OFAction> ofActions = new ArrayList<OFAction>();
        ofActions.add(ofAction);

        // Send this flow modification message to the switch.
        sendFlowModMessage( ofSwitch,
                            OFFlowMod.OFPFC_ADD,
                            ofMatch,
                            ofActions,
                            PRIORITY_IP_PACKETS,
                            NO_IDLE_TIMEOUT,
                            BUFFER_ID_NONE );
    }


    //- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    private void setArpFlow( final IOFSwitch ofSwitch )
    {
        // Create match object match arp packets
        OFMatch ofMatch = new OFMatch();
        ofMatch.setWildcards( allExclude(OFPFW_DL_TYPE) )
                            .setDataLayerType( Ethernet.TYPE_ARP );

        // Create output action to forward normally
        OFActionOutput ofAction  = new OFActionOutput(
                                    OFPort.OFPP_NORMAL.getValue(),
                                    (short) 65535               );

        // Create our action list and add this action to it.
        List<OFAction> ofActions = new ArrayList<OFAction>();
        ofActions.add(ofAction);

        // Send this flow modification message to the switch.
        sendFlowModMessage( ofSwitch,
                            OFFlowMod.OFPFC_ADD,
                            ofMatch,
                            ofActions,
                            PRIORITY_ARP_PACKETS,
                            NO_IDLE_TIMEOUT,
                            BUFFER_ID_NONE );
    }

}
```