B.Tech IT 6th SEMESTER MINI PROJECT

Technical Report : Implementation of Load Balancer over SDN using Floodlight.

Team Members

Somil Gupta, Utkarsh Gaur

Project Mentor

Dr. Mayank Pandey

Background: Loadbalancer is a network control module which helps in reducing the traffic/load on a single resource. It uses the technique of load redirection by making the clients interact with a particular Virtual IP which is resolved into one of the servers present in the virtual IP pool. Although this technique is being utilised in traditional systems by the use of anycast IPs or in DNS resolution, we have tried to implement the same via SDN technology with the controller performing the load balancing. Besides, We have tried to enhance the load balancing latency by setting a proxy ARP module to answer ARP queries and prevent the costly ARP broadcast.

Approach: Implementation of the load balancer module using centralised controller has been accomplished in Floodlight.

1. After the topology module has created the topology and device manager module has recognised the devices in the topology, incoming packet is analysed by the load balancer module to know the source and destination addresses.
2. A queue is maintained of all the servers that corresponds to a particular virtual IP. *Circular rotation* is technique is followed to balance the load among various servers.
3. If the destination points to one of the virtual IPs notified in the module (by the network administrator), it is directed to the first server in the queue. A route between client and server is found and flow entries are added in all the switches in the route.(using routingservice and ForwardingBase).
4. Thereafter any further requests from the client are handled by that server. Only when the flow entries expire or a request from a new device arrives, does the controller create a route for the next queued server in the route.
5. When packets are returned from the servers, care is taken to convert the source IP back to virtual IP. In this manner we are hiding the servers from the client who feels he is being answered by a single system.
6. Apart from load balancing, the controller also performs Proxy Address Resolution. Since the controller is already aware of the devices, their IPs and their MAC addresses, any ARP request that comes to the controller is answered by the controller itself and no flow entries are added or broadcasts are done. It also handles the answering of ARP requests for virtual IPs.

Implementation:

ServerLoadBalance Module

```
public class AdvancedLoadBalancer  extends ForwardingBase implements
            IFloodlightModule,IOFMessageListener{
      protected static Queue<Long> ServerDevicesList;
      protected static HashMap<Long,IPv4Address> Device2IP;
      protected static Vector<IPv4Address> Servers;
      protected static IPv4Address VirtualServerIP;
      protected static MacAddress VirtualServerMAC;
```

- **Dependencies:**
  - IFloodlightProviderService for registrating as a module.
  - IDeviceService for finding the source and destination devices and servers.
  - ITopologyService for extracting overlay topology.
  - IRoutingService for getting the route between source and destination.
- **Pre Required Modules:** Device Manager, topology and proxyarp modules.
- **Post Required Modules:** Forwarding Module.
- **Listeners Involved:**
  - IOFMessageListener for being notified when a packet_in request arrives.
  - IDeviceListener (in the ServersRetriever Class) for being notified when a device is added or the status of a device changes.
- **BaseClass** ForwardingBase to utilise the pushRoute() method which pushes a particular flow entry on all the switches between source and destination in the route. Also to make use of default configuration parameters.
- **Data Members:**
  - **`Queue<Long>`** ***`ServerDevicesList`*****`;`** maintains the queue of the servers that resolve the requested virtual IP. Elements include the device key of the servers present.
  - `HashMap<Long,IPv4Address>` *`Device2IP`*`;` maintains the hash map of device key to the corresponding IP address only for the servers.
  - `Vector<IPv4Address>` *`Servers`*`;` contains the IP addresses of the servers provided by the network administrator.
  - `IPv4Address` *`VirtualServerIP`*`;` it is the virtual ip of the virtual server.
  - `MacAddress` *`VirtualServerMAC`*`;` it is the vitual MAC address of the virtual server.
- **Data Functions:**
  - *AdvanceLoadBalancer.java*
    - ***`public void`*** *`setServerandVIP()`* dummy function where the servers vector and VirtualServer IP and MAC are initialised. This is to prevent the network administrator to go through the code unnecessarily.
    - ***`protected`*** `Match createMatchFromPacket((IOFSwitch sw, OFPort inPort, FloodlightContext cntx)` Taken from forwarding module. It creates a generic match based upon packet received and the signal of the flags *`FLOWMOD_DEFAULT_MATCH_VLAN`*`,`*`FLOWMOD_DEFAULT_MATCH_MAC`*`,` *`FLOWMOD_DEFAULT_MATCH_IP_ADDR`*`,` *`FLOWMOD_DEFAULT_MATCH_TRANSPORT`*`.`The function therefore helps to reuse the basic match building functionality to add in further modification on our own.
    - **`protected`** `Route createRoute(IDevice srcDevice, IDevice dstDevice,IOFSwitch sw, OFPort inPort):` First checks whether not source and server not the same device, then it checks whether source and destination both lie at the boundary of the topology(prevent answering for a packet out by a switch). Finally when it is ensured that both devices do have valid attachment points, does it use the getRoute() function in IroutingService to get the shortest path. Since the getroute() function does not work when both the devices are on the same switch, in that case it creates a route on its own and returns the route.
    - **`protected`** `Command doForwardFlow(IOFSwitch sw, OFPacketIn pi, FloodlightContext cntx,`**`boolean`**` dir)` It extracts the source and the destination devices from the context(already found by the device manager module). Implements the main logic of the forwarding as described later.
    - **`public`** `net.floodlightcontroller.core.Ilistener.Command processPacketInMessage(...):` Filters out Ipv4 packets which either are destined to virtual ip or coming from one of the servers ip.For all others, it sends the control to the next module.

- **public** net.floodlightcontroller.core.IListener.Command receive Function that is triggered when a packet is received. Fiters out packet_in requests which have forwarding decision as FORWARD or FORWARD_OR_FLOOD and for all others command.continue.

- *ServersRetriever.java*
  - Implements IdeviceListener. An instance of it is created in the startup of the module and registered with the IdeviceListener.
  - **public void** deviceAdded(IDevice device) Whenver a device is added it checks whether the IP corresponds with any of the servers. If yes, it is added to the queue and the hashmap.
  - **public void** deviceRemoved(IDevice device) If the device that is removed is a server then the entry is also removed from the data structures.
  - **public void** deviceMoved(IDevice device) Called when the port goes down. If the port connected to a server goes down then it is removed from the data structures.

ProxyArpModule

```
public class ProxyArp  implements IOFMessageListener, IFloodlightModule {
      HashMap<IPv4Address,MacAddress> vmac;
      protected static final long BROADCAST_MAC = 0xffffffffffffL;
```

- **Dependencies:** Same as above
- **Pre Required Modules:** Device Manager, topology.
- **Post Required Modules:** Forwarding Module.
- **Listeners Involved:** IOFMessageListener.
- **Internal Class:** ARPRequest(SourceMAC address, SourceIP address, TargerMAC address, TargetIP address, switchid, inport): Encapsulates ARP request.
- **Data Members:**
  - **HashMap<IPv4Address,MacAddress> vmac;** HashMap of virtual Ips with their MACs.
- **Data Functions:**
  - **public net.floodlightcontroller.core.Ilistener.Command processPacketInMessage(...):** Filter out ARP request packets which are not gratituious and send handle ARP request. For all other non-ARP packets, command.continue else stop the execution. As their is no need to send the ARP packets further.
  - **public net.floodlightcontroller.core.IListener.Command receive** Same as in above module.
  - **protected** Command handleARPRequest(ARP arp, DatapathId switchId, OFPort portId, FloodlightContext cntx) Using the query devices function of IdevicesService, the device correspnding to target ip is found. If device is not found then it searches for the IP in the vmac HashMap and sends the corresponding virtual MAC. If still not found then it is forwarded to forwarding modulefor broadcast. Request forward to sendARPReply().
  - **protected void** sendARPReply(ARPRequest arpRequest) creates an ARP reply packet with the obtained IP address and sends PO message for the packet_out.
  - **protected void** sendPOMessage(IPacket packet, IOFSwitch sw, OFPort port) sends the packet_out through the inputport.

Forwarding Logic: (Using pushRoute() and getRoute() implementation**)**

- **If dir=true i.e from client to a virtual ip**, A route is found from the source to the destination and packets with the following match,action are inserted in all the intermediate switches.
    - **Match :** src_ip=client ip. dst_ip=server ip, src_mac=client mac, dst_mac=server mac, src_port and dst_port=dont care, inport are set by the pushRoute function itself**.**
    - **Action:** set the outport as per the route done by pushRoute function.
    - Additionally, at the first switch a flow entry is added with the following match-action pair.
    - **Match :** src_ip=client ip. dst_ip=virtual ip, src_mac=client mac, dst_mac=virtual mac, src_port and dst_port=dont care, inport is device attachment port.
    - **Action:** set dst_ip=server ip, dst_mac=server mac, outport using the snippet below.

```
OFPort outPort= route.getPath().get(1).getPortId();
```

A packet out is additionally sent by the pushRoute which is received by the successive switch and ignored by the controller.

- **If dir=false i.e from server to client ip** , A route is found from the source to the destination and packets with the following match,action are inserted in all the intermediate switches.
    - **Match :** src_ip=virtual ip, dst_ip=client ip, src_mac=virtual mac, dst_mac=client mac, src_port and dst_port=dont care, inport are set by the pushRoute function itself**.**
    - **Action:** set the outport as per the route done by pushRoute function.

The getRoute function was sent actual client and server while the match was added for the virtual ip.

- Additionally, at the first switch a flow entry is added with the following match-action pair.
- **Match :** src_ip=server ip. dst_ip=client ip, src_mac=server mac, dst_mac=client mac, src_port and dst_port=dont care, inport is server attachment port.
- **Action:** set src_ip= virtual ip, src_mac=virtual mac, outport using the snippet above.

A packet out is additionally sent by the pushRoute which is received by the successive switch and ignored by the controller.

- **NOTE: It is important either to stop the control after this or increase the priority of reverse flow entry at the switch near to destination, otherwise the flow entry with three actions gets overwritten.**

Improvements introduced:

- **Code reuse and modularity** by making use of implemented getRoute and pushRoute and making ProxyArp as separate module.
- **Decoupling from the transport layer**. In this manner a change of port does not need an additional flow entry in the switches.
- **Stoppage of unnecessary broadcasts and ARP related flow entries** when the controller has the information regarding the devices present in the topology.
- ServersRetriever **keeps updating the data structures** when a device changes its status so instead of maintaining IP addresses in the queue which need to be resolved when assigned(via loopover all the devices), our structures rather maintain device keys which allow for **faster device resolution approach.**
- **No disturbance or change of code** has been made to forwarding base module.
- **A generic match build function** allows for more cleaner code, used from forwarding module.

Shortcomings still untackled:

- **Mobility problem**: When any device is removed or port down status is observed, flow entries are not deletedfrom the switches. When a device comes back again i.e. port up occurs, due to no-trigger of any function in IdeviceListener the device is not registered in data structures.
- Works in a subnet, g**ateway functionality not taken care**. Although virtual ip facility is provided which will help to implement gateway functionality in future.
- Only works with a **single Vip pool** at present.
- The proxyArp module optimises on the condition that the **devices have sent gratituous ARPs** on attachment.